



PERFORMANCE EVALUATION OF A FIELD PROGRAMMABLE  
GATE ARRAY-BASED SYSTEM FOR DETECTING AND TRACKING  
PEER-TO-PEER PROTOCOLS ON A GIGABIT ETHERNET NETWORK

THESIS

Brennon D. Thomas

AFIT/GCO/ENG/10-20

DEPARTMENT OF THE AIR FORCE  
AIR UNIVERSITY

**AIR FORCE INSTITUTE OF TECHNOLOGY**

Wright-Patterson Air Force Base, Ohio

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the United States Government.

AFIT/GCO/ENG/10-20

PERFORMANCE EVALUATION OF A FIELD PROGRAMMABLE  
GATE ARRAY-BASED SYSTEM FOR DETECTING AND TRACKING  
PEER-TO-PEER PROTOCOLS ON A GIGABIT ETHERNET NETWORK

THESIS

Presented to the Faculty  
Department of Electrical and Computer Engineering  
Graduate School of Engineering and Management  
Air Force Institute of Technology  
Air University  
Air Education and Training Command  
In Partial Fulfillment of the Requirements for the  
Degree of Master of Science

Brennon D. Thomas, BSEE

June 2010

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

AFIT/GCO/ENG/10-20

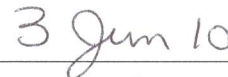
PERFORMANCE EVALUATION OF A FIELD PROGRAMMABLE  
GATE ARRAY-BASED SYSTEM FOR DETECTING AND TRACKING  
PEER-TO-PEER PROTOCOLS ON A GIGABIT ETHERNET NETWORK

Brennon D. Thomas, BSEE

Approved:



Dr. Barry E. Mullins (Chairman)



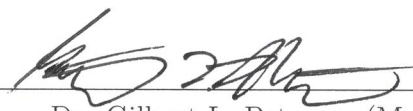
date



Dr. Robert F. Mills (Member)



date



Dr. Gilbert L. Peterson (Member)



date

## *Abstract*

Recent years have seen a massive increase in illegal, suspicious, and malicious traffic traversing government and military computer networks. Some examples include illegal file distribution and disclosure of sensitive information using the BitTorrent file sharing protocol, criminals and terrorists using Voice over Internet Protocol (VoIP) technologies to communicate, and foreign entities exfiltrating sensitive data from government, military, and Department of Defense contractor networks.

As a result of these growing threats, the TRacking and Analysis for Peer-to-Peer (TRAPP) system was developed in 2008 to detect BitTorrent and VoIP traffic of interest. The TRAPP system, designed on a Xilinx Virtex-II Pro Field Programmable Gate Array (FPGA) proved valuable and effective in detecting traffic of interest on a 100 Mbps network. Using concepts and technology developed for the TRAPP system, the TRAPP-2 system is developed on a Xilinx ML510 FPGA. The goals of this research are to evaluate the performance of the TRAPP-2 system as a solution to detect and track malicious packets traversing a gigabit Ethernet network. The TRAPP-2 system detects a BitTorrent, Session Initiation Protocol (SIP), or Domain Name System (DNS) packet, extracts the payload, compares the data against a hash list, and if the packet is suspicious, logs the entire packet for future analysis.

Results show that the TRAPP-2 system captures 95.56% of BitTorrent, 20.78% of SIP INVITE, 37.11% of SIP BYE, and 91.89% of DNS packets of interest while under a 93.7% network utilization (937 Mbps). For another experiment, the contraband hash list size is increased from 1,000 to 131,072,000 unique items. The experiment reveals that each doubling of the hash list size results in a mean increase of approximately 16 central processing unit cycles. These results demonstrate the TRAPP-2 system's ability to detect traffic of interest under a saturated network utilization while maintaining large contraband hash lists.

## *Acknowledgements*

First and foremost, I would like to thank God for seeing me through this process and for continuing to bless me in my life.

Many thanks to my advisor, Dr. Mullins, for his advice and insight while writing and researching this thesis. I would also like to thank my committee members, Dr. Mills and Dr. Peterson, for their support and suggestions.

I would not be where I am today if it was not for the love and guidance of my parents. Thank you for raising me to be the man I am today.

Lastly, I want to thank my wife for her unending love, support, and understanding throughout my time here at AFIT. None of this would have been possible without you.

Brennon D. Thomas

# *Table of Contents*

	Page
Abstract . . . . .	iv
Acknowledgements . . . . .	v
List of Figures . . . . .	x
List of Tables . . . . .	xiii
List of Abbreviations . . . . .	xv
 I. Introduction . . . . .	 1
1.1 Motivation . . . . .	1
1.2 Overview and Goals . . . . .	1
1.3 Thesis Layout . . . . .	2
 II. Literature Review . . . . .	 4
2.1 Illicit Traffic . . . . .	4
2.1.1 The BitTorrent Protocol . . . . .	4
2.1.2 The Voice over Internet Protocol . . . . .	7
2.1.3 The Domain Name System . . . . .	10
2.2 Analyzing and Classifying Network Traffic . . . . .	15
2.2.1 Port Matching Analysis . . . . .	15
2.2.2 Payload Analysis . . . . .	16
2.2.3 Behavioral Analysis . . . . .	17
2.3 Network Traffic and Data Obfuscation Methods . . . . .	18
2.3.1 Byte Padding . . . . .	18
2.3.2 Ron's Code 4 . . . . .	18
2.3.3 Virtual Private Network and Secure Shell Tunnels . . . . .	19
2.3.4 Darknets . . . . .	20
2.3.5 The Onion Router Network . . . . .	21
2.4 Current Methods for Detecting Illicit Traffic . . . . .	21
2.4.1 Wireshark . . . . .	21
2.4.2 Snort . . . . .	22
2.4.3 Hi-Performance Protocol Identification Engine . . . . .	22
2.4.4 BitTorrent Monitoring System . . . . .	23
2.4.5 Entropy-Based Malicious DNS detection . . . . .	25
2.4.6 Cross Entropy-Based Malicious DNS detection . . . . .	26
2.4.7 Detecting DNS Tunnels Using Artificial Neural Networks . . . . .	26

	Page
2.5 The Substitute Database Manager Hashing Function . .	27
2.6 The Tracking and Analysis for Peer-to-Peer System . . .	28
2.6.1 Capabilities . . . . .	29
2.6.2 Limitations . . . . .	29
2.7 Summary . . . . .	31
III. Methodology . . . . .	32
3.1 Goals and Hypotheses . . . . .	32
3.2 Approach . . . . .	34
3.2.1 Hardware Modifications . . . . .	34
3.2.2 Software Modifications . . . . .	35
3.2.3 Algorithm . . . . .	36
3.3 System Boundaries . . . . .	38
3.4 System Services . . . . .	39
3.5 Workload . . . . .	40
3.5.1 Packet Workload Characteristics . . . . .	41
3.5.2 BitTorrent Workload . . . . .	42
3.5.3 SIP Workload . . . . .	44
3.5.4 DNS Workload . . . . .	49
3.5.5 Non-BitTorrent/SIP/DNS Workload . . . . .	52
3.5.6 Network Load . . . . .	53
3.6 Performance Metrics . . . . .	53
3.6.1 Packet Processing Time . . . . .	54
3.6.2 Probability of Packet Intercept . . . . .	54
3.6.3 Network Utilization . . . . .	54
3.7 System Parameters . . . . .	56
3.8 Factors . . . . .	56
3.9 Evaluation Technique . . . . .	58
3.9.1 Calculating Packet Processing Time . . . . .	60
3.9.2 Calculating Probability of Packet Intercept . . .	60
3.10 Experimental Design . . . . .	61
3.10.1 Experiment 1 . . . . .	61
3.10.2 Experiment 2 . . . . .	61
3.10.3 Experiment 3 . . . . .	62
3.10.4 Experiment 4 . . . . .	62
3.11 Methodology Summary . . . . .	62



	Page
IV. Results and Analysis . . . . .	64
4.1 Results and Analysis of Experiment 1 . . . . .	64
4.1.1 BitTorrent Packet Processing Time . . . . .	65
4.1.2 SIP Packet Processing Time . . . . .	66
4.1.3 DNS Packet Processing Time . . . . .	67
4.1.4 Experiment 1 Analysis . . . . .	68
4.2 Results and Analysis of Experiment 2 . . . . .	69
4.2.1 Experiment 2 Analysis . . . . .	70
4.3 Results and Analysis of Experiment 3 . . . . .	71
4.3.1 BitTorrent Probability of Packet Intercept . . . . .	71
4.3.2 SIP INVITE Probability of Packet Intercept . . . . .	72
4.3.3 SIP BYE Probability of Packet Intercept . . . . .	74
4.3.4 DNS Probability of Packet Intercept . . . . .	75
4.3.5 Experiment 3 Analysis . . . . .	77
4.4 Results and Analysis of Experiment 4 . . . . .	78
4.4.1 Experiment 4 Analysis . . . . .	81
4.5 Overall Analysis . . . . .	82
4.6 Summary . . . . .	83
V. Conclusions . . . . .	84
5.1 Conclusions of Research . . . . .	84
5.1.1 Goal #1: Determine the packet processing times for packets of interest . . . . .	84
5.1.2 Goal #2: Determine the probability of packet in- tercept under a flood of 400 packets of interest . . . . .	84
5.1.3 Goal #3: Determine the probability of packet in- tercept under various network utilizations . . . . .	84
5.1.4 Goal #4: Determine how the hash list size affects packet processing time . . . . .	85
5.2 Significance of Research . . . . .	85
5.3 Recommendations for Future Research . . . . .	86
Appendix A. Experimental Data . . . . .	88
A.1 Results of Experiment 1 . . . . .	88
A.2 Results of Experiment 2 . . . . .	92
A.3 Results of Experiment 3 . . . . .	93
A.4 Results of Experiment 4 . . . . .	98

	Page
Appendix B. Pilot Test Data . . . . .	103
B.1 Results of Pilot Studies . . . . .	103
B.1.1 BRAM versus SDRAM Memory Scheme . . . . .	103
B.1.2 DNS Packet Detection . . . . .	105
B.1.3 sdbm Hashing Times . . . . .	106
B.1.4 Packet Size Transfer Times . . . . .	108
Appendix C. Constructing the TRAPP-2 System Hardware . . . . .	110
C.1 Hardware Description . . . . .	110
C.1.1 Microprocessor . . . . .	110
C.1.2 Synchronous Dynamic Random Access Memory	110
C.1.3 Block Random Access Memory . . . . .	110
C.1.4 XPS Hard Ethernet Media Access Controller . .	110
C.1.5 RS232 Universal Asynchronous Receiver/Trans-	111
mitter . . . . .	
C.1.6 XPS Timer . . . . .	111
C.2 Component Configuration . . . . .	111
C.3 Converting from MII to RGMII v2.0 . . . . .	124
Bibliography . . . . .	128
Vita . . . . .	132

## *List of Figures*

Figure		Page
2.1.	The Session Initiation Protocol Process [Cis10]. . . . .	9
2.2.	The Domain Name System Distributed Database [Moh09]. . . . .	10
2.3.	Establishing a Domain Name System Tunnel. . . . .	13
2.4.	The BitTorrent Monitoring System Process [CCM <sup>+</sup> 07]. . . . .	24
2.5.	Entropy Changes In the Domain Name System External Query Traffic [RKSM08]. . . . .	25
2.6.	TRacking and Analysis for Peer-to-Peer System Flowchart [Sch09].	30
3.1.	Summary of Experiments for the TRacking and Analysis for Peer-to-Peer 2 System. . . . .	33
3.2.	Packet Data Flow in the TRacking and Analysis for Peer-to-Peer 2 System. . . . .	37
3.3.	The TRacking and Analysis for Peer-to-Peer 2 System Under Test.	38
3.4.	BitTorrent Packet Type Hierarchy for the TRacking and Analysis for Peer-to-Peer 2 System. . . . .	42
3.5.	Session Initiation Protocol Packet Type Hierarchy for the TRacking and Analysis for Peer-to-Peer 2 System. . . . .	44
3.6.	Domain Name System Packet Type Hierarchy for the TRacking and Analysis for Peer-to-Peer 2 System. . . . .	50
3.7.	Packet Creation and Experimental Hardware Configuration Setup for the TRacking and Analysis for Peer-to-Peer 2 System. . . . .	59
3.8.	Experimental Setup for the TRacking and Analysis for Peer-to-Peer 2 System. . . . .	60
4.1.	Mean Packet Processing Times for the 12 Different Packet Types for Experiment 1. . . . .	65
4.2.	Mean Packet Processing Times for BitTorrent Packet Types for Experiment 1. . . . .	66
4.3.	Mean Packet Processing Times for Session Initiation Protocol Packet Types for Experiment 1. . . . .	67

Figure		Page
4.4.	Mean Packet Processing Times for Domain Name System Packet Types for Experiment 1. . . . .	68
4.5.	Network Utilization and Probability of Packet Intercept vs Flood of 1200 packets (400 packets x 3 replications) Worst-Case Scenario Packets for Experiment 2. . . . .	70
4.6.	Probability of Packet Intercept for BitTorrent Packets vs Various Network Utilizations for Experiment 3. . . . .	72
4.7.	Probability of Packet Intercept for Session Initiation Protocol INVITE Packets vs Various Network Utilizations for Experiment 3. . . . .	74
4.8.	Probability of Packet Intercept for Session Initiation Protocol BYE Packets vs Various Network Utilizations for Experiment 3. . . . .	75
4.9.	Probability of Packet Intercept for Domain Name System Packets vs Various Network Utilizations for Experiment 3. . . . .	77
4.10.	Mean Packet Processing Times vs 17 Different Hash List Sizes for Experiment 4. . . . .	79
4.11.	Mean Packet Processing Times vs Natural Log of 17 Different Hash List Sizes for Experiment 4. . . . .	80
C.1.	The Project Creation Options Window. . . . .	111
C.2.	The Project Creation and Repository Selection Window. . . . .	112
C.3.	The Base System Builder Design Selection Window. . . . .	113
C.4.	The Board Selection Window. . . . .	114
C.5.	The Processor Selection Window. . . . .	115
C.6.	The Processor Configuration Window. . . . .	116
C.7.	The Peripheral Configuration Window. . . . .	117
C.8.	The Processor Cache Configuration Window. . . . .	118
C.9.	The Application Selection Window. . . . .	119
C.10.	The Summary Configuration Window. . . . .	120
C.11.	The Configure Libraries and Drivers Window. . . . .	121
C.12.	The Software Platform Settings Window. . . . .	122

Figure		Page
C.13.	The Software Platform Settings OS and Lib Configuration Window. . . . .	123

## *List of Tables*

Table		Page
2.1.	Speed Comparisons of Symmetric Ciphers on a Pentium II [Sta06].	19
3.1.	BitTorrent Packet Weights for the TRacking and Analysis for Peer-to-Peer 2 System. . . . .	43
3.2.	Session Initiation Protocol INVITE Packet Weights for the TRacking and Analysis for Peer-to-Peer 2 System. . . . .	45
3.3.	Session Initiation Protocol BYE Packet Weights for the TRacking and Analysis for Peer-to-Peer 2 System. . . . .	46
3.4.	Domain Name System Packet Weights for the TRacking and Analysis for Peer-to-Peer 2 System. . . . .	50
3.5.	Network Utilizations Due to the Linux pktgen Utility Load in the TRacking and Analysis for Peer-to-Peer 2 System. . . . .	53
3.6.	Factor Levels for Experiments 1, 2, and 3, for the TRacking and Analysis for Peer-to-Peer 2 System. . . . .	57
3.7.	Factor Levels for Experiment 4 for the TRacking and Analysis for Peer-to-Peer 2 System. . . . .	58
4.1.	Sorted Mean Packet Processing Times for Experiment 1. . . . .	64
4.2.	Sorted BitTorrent Mean Packet Processing Times for Experiment 1. . . . .	65
4.3.	Sorted Session Initiation Protocol Mean Packet Processing Times for Experiment 1. . . . .	66
4.4.	Sorted Domain Name System Mean Packet Processing Times for Experiment 1. . . . .	67
4.5.	Probability of Packet Intercept for Flood of 1200 (400 packets x 3 replications) Worst-Case Scenario Packets for Experiment 2. .	69
4.6.	Probability of Packet Intercept for BitTorrent Packets Under Various Network Utilizations for Experiment 3. . . . .	71
4.7.	Probability of Packet Intercept for Session Initiation Protocol INVITE Packets Under Various Network Utilizations for Experiment 3. . . . .	73

Table		Page
4.8.	Probability of Packet Intercept for Session Initiation Protocol BYE Packets Under Various Network Utilizations for Experiment 3. . . . .	74
4.9.	Probability of Packet Intercept for Domain Name System Packets Under Various Network Utilizations for Experiment 3. . . . .	76
4.10.	Mean Packet Processing Times for 17 Different Hash List Sizes for Experiment 4. . . . .	78
4.11.	Difference Between Mean Packet Processing Times for 17 Differ- ent Hash List Sizes for Experiment 4. . . . .	81
A.1.	CPU Cycle Data for Experiment 1. . . . .	89
A.2.	CPU Cycle Data for Experiment 1 Continued. . . . .	90
A.3.	CPU Cycle Data for Experiment 1 Continued. . . . .	91
A.4.	Packets Captured for Experiment 2. . . . .	92
A.5.	Packets Captured for Experiment 3, Utilization 1 ( $\approx 20.4\%$ ). . .	93
A.6.	Packets Captured for Experiment 3, Utilization 2 ( $\approx 30.1\%$ ). . .	94
A.7.	Packets Captured for Experiment 3, Utilization 3 ( $\approx 40.8\%$ ). . .	94
A.8.	Packets Captured for Experiment 3, Utilization 4 ( $\approx 49.8\%$ ). . .	95
A.9.	Packets Captured for Experiment 3, Utilization 5 ( $\approx 60.2\%$ ). . .	95
A.10.	Packets Captured for Experiment 3, Utilization 6 ( $\approx 71.4\%$ ). . .	96
A.11.	Packets Captured for Experiment 3, Utilization 7 ( $\approx 81.8\%$ ). . .	96
A.12.	Packets Captured for Experiment 3, Utilization 8 ( $\approx 93.7\%$ ). . .	97
A.13.	CPU Cycle Data for Experiment 4. . . . .	98
A.14.	CPU Cycle Data for Experiment 4 Continued. . . . .	99
A.15.	CPU Cycle Data for Experiment 4 Continued. . . . .	100
A.16.	CPU Cycle Data for Experiment 4 Continued. . . . .	101
A.17.	CPU Cycle Data for Experiment 4 Continued. . . . .	102
B.1.	CPU Cycles Used to Process a SIP Packet. . . . .	104
B.2.	CPU Cycles Used to Identify a DNS Packet. . . . .	105
B.3.	CPU Cycles Used to sdbm hash a SIP Packet. . . . .	107
B.4.	CPU Cycles Used to Copy Smallest versus Largest Packet. . . .	109

## *List of Abbreviations*

Abbreviation		Page
VoIP	Voice over Internet Protocol . . . . .	1
DNS	Domain Name System . . . . .	1
TRAPP	TRacking and Analysis for Peer-to-Peer . . . . .	1
FPGA	Field Programmable Gate Array . . . . .	1
SIP	Session Initiation Protocol . . . . .	2
CPU	Central Processing Unit . . . . .	2
sdbm	Substitute Database Manager . . . . .	2
HTTP	Hypertext Transfer Protocol . . . . .	5
TCP	Transmission Control Protocol . . . . .	6
SHA-1	Secure Hash Algorithm 1 . . . . .	6
IP	Internet Protocol . . . . .	8
URI	Uniform Resource Identifier . . . . .	9
UDP	User Datagram Protocol . . . . .	12
MD5	Message-Digest algorithm 5 . . . . .	15
VPN	Virtual Private Network . . . . .	18
SSH	Secure Shell . . . . .	18
HiPPiE	Hi-Performance Protocol Identification Engine . . . . .	22
LAN	Local Area Network . . . . .	29
SPAN	Switched Port Analyzer . . . . .	29
BRAM	Block Random Access Memory . . . . .	34
SDRAM	Synchronous Dynamic Random Access Memory . . . . .	35
SUT	System Under Test . . . . .	38
CUT	Component Under Test . . . . .	38
BASH	Bourne Again SHell . . . . .	55



# PERFORMANCE EVALUATION OF A FIELD PROGRAMMABLE GATE ARRAY-BASED SYSTEM FOR DETECTING AND TRACKING PEER-TO-PEER PROTOCOLS ON A GIGABIT ETHERNET NETWORK

## I. Introduction

### 1.1 *Motivation*

Billions of packets traverse government and military networks every day. Often, these packets have legitimate destinations. Unfortunately, the past few years have seen a massive increase in illegal, suspicious, and malicious traffic. Some examples include BitTorrent illegal file distribution, suspects of interest using Voice over Internet Protocol (VoIP) phones to conduct business, and Domain Name System (DNS) data exfiltration. Recent stories include blueprints for Marine One being leaked by a United States contractor using a BitTorrent file sharing program, the Mumbai terrorists using VoIP phones to communicate, and Chinese hackers pilfering intellectual property from Google and other United States companies [FOX09] [Kah08] [Wir10].

As a result of these growing threats, the TRacking and Analysis for Peer-to-Peer (TRAPP) system was developed to detect BitTorrent and VoIP traffic of interest [Sch09]. The system resides on a Xilinx Virtex-II Pro Field Programmable Gate Array (FPGA). The first iteration prototype is limited in both processing speed and by a 100 megabit Ethernet card, but still captures packets of interest with a “probability of intercept of at least 99.0%, using a 95% confidence interval and given an 89.6 Mbps network utilization” [Sch09]. These results prove the TRAPP system is a viable tool worth expanding its capabilities to detect malicious network traffic.

### 1.2 *Overview and Goals*

This research extends the technology and concepts of the first TRAPP system by implementing a more powerful FPGA and incorporating an additional protocol. The

focus of this research is create a second generation TRAPP system, named TRAPP-2, that is designed on a Xilinx ML510 FPGA board with a faster processor and a gigabit Ethernet controller [Xil09]. The original TRAPP system detects the BitTorrent and Session Initiation Protocol (SIP) peer-to-peer protocols in real-time. For the TRAPP-2 system, malicious Domain Name System (DNS) detection is added. Ultimately, the research determines that the TRAPP-2 system is a feasible solution to detect and track protocols of interest for law enforcement and intelligence agencies on gigabit Ethernet networks.

The TRAPP-2 system meets four measurement goals. The first goal determines the packet processing times for packets detected by the TRAPP-2 system. The second goal determines the probability of packet intercept under a flood of packet-of-interest traffic. The third goal determines the probability of packet intercept under various network utilizations. The last goal determines how increasing the hash list size affects the packet processing time. The two metrics used to measure performance are packet processing time, measured in Central Processing Unit (CPU) cycles, and probability of packet intercept.

### ***1.3 Thesis Layout***

Chapter 1 outlines the motivation, overview, and goals of the research. An overview, background information, and related research on illicit traffic, network traffic classification, network traffic obfuscation methods, current methods of detecting malicious and illegal network traffic, the Substitute Database Manager (sdbm) hashing function, and the TRAPP system are covered in Chapter 2. Chapter 3 explains the method and experiments used to evaluate the performance of the TRAPP-2 system. Chapter 4 presents the analysis and discussion of the results from the experiments. The conclusions drawn from the research, real-world significance, and future research areas are detailed in Chapter 5. Appendix A contains all of the experimental data. Appendix B contains the pilot test data used to design and build the TRAPP-2 sys-

tem. Appendix C provides a hardware construction guide to build the TRAPP-2 system.

## II. Literature Review

This chapter presents an overview, background information, and related research on illicit traffic, network traffic classification, network traffic obfuscation methods, current methods of detecting malicious and illegal network traffic, the Substitute Database Manager (sdbm) hashing function, and the TRacking and Analysis for Peer-to-Peer (TRAPP) system. Section 2.1 provides a brief overview of the BitTorrent, Voice over Internet Protocol (VoIP), and Domain Name System (DNS) protocols and their illegitimate uses. The methods of classifying network traffic are detailed in Section 2.2. Section 2.3 covers some of the obfuscation and evasion methods used to hide network traffic and data. This allows for an exploration of the current methods of detecting illicit traffic in Section 2.4. This is followed by an examination of the sdbm hashing function in Section 2.5 and the current TRAPP system’s capabilities and limitations in Section 2.6. The chapter is summarized in Section 2.7.

### 2.1 *Illicit Traffic*

The Internet has evolved from a small network of sparsely connected computers to an expansive web of millions. The rapid access to information, knowledge, and current events has been paralleled with a proliferation of illicit data and traffic. BitTorrent, VoIP, and DNS are legitimate protocols and services; however, they can also be used for illicit purposes. Some examples include the distribution of illegal files using BitTorrent, terrorists using VoIP for command and control during operations, and hackers exploiting DNS to exfiltrate sensitive data from networks [Kah08] [Van09].

*2.1.1 The BitTorrent Protocol.* The BitTorrent protocol is the natural evolution of file sharing protocols [Coh08]. The BitTorrent protocol was created by Bram Cohen as an alternative to the centralized file sharing programs such as Napster and Gnutella [Coh08].

The Napster file sharing program was created by Shawn Fanning to allow his friends to share and distribute .mp3 music files [Tys08]. The popularity of Napster

exploded and became the preferred method of sharing legal and illegal music files over the Internet. The Napster system relied on a central server, run by the Napster organization, to point clients to the specific .mp3 music files requested. The server acted as a mediator to set up the direct peer-to-peer connection between the file downloader and the file uploader. The servers did not store or host any of the actual .mp3 files being downloaded. It did not take long for record companies and prosecutors to target Napster for the dissemination of copyrighted music. The central servers that the Napster system relied on proved to be an easy target for the Recording Industry Association of America and various music labels. Ultimately, the lawsuits forced Napster to shut down in 2001 [Fel04].

As a result of the Napster shutdown, file sharing programs migrated toward a decentralized approach. This eliminated the need for a centralized server and led to the development of BitTorrent [Coh08]. In addition to a decentralized architecture, the BitTorrent protocol implemented two new methods of downloading files. The first method was to break the file into blocks of 256 kilobytes. This allowed downloaders to accumulate different blocks, or parts of the file, and assemble them upon download completion to create the entire file. As soon as a block was completely downloaded, it was immediately uploaded to other peers seeking the file. To aid in the speed of downloading, the BitTorrent protocol was designed to capitalize on the disparate download versus upload speeds offered by Internet Service Providers (ISPs). The downloader of a file was able to simultaneously download blocks of the file from different uploaders. Since ISPs provided download speeds significantly greater than upload speeds, a downloader could accumulate numerous smaller peer upload speeds to match his download speed [Coh08]. Over time, the BitTorrent protocol has become the preferred method of sharing files over the Internet due to its efficiency.

Currently, the BitTorrent protocol consists of two different protocols. The first is the BitTorrent Tracker protocol which runs over the Hypertext Transfer Protocol (HTTP). It communicates between clients and a tracker website to point clients to

the peers sharing the requested file. A tracker website maintains a dynamic database of peers associated with a file [Coh08].

The second, and more relevant protocol for this research, is the peer wire protocol. The peer wire protocol runs over the Transmission Control Protocol (TCP) and is used to exchange the file pieces specified in the file's .torrent file. The peer wire protocol relies on the Secure Hash Algorithm 1 (SHA-1) for file identification and data block integrity verification [Coh08]. SHA-1, outlined in Request for Comments (RFC) 3174, is a United States Government algorithm formally named as the Federal Information Processing Standards Publication 180-1 (FIPS 180-1) [RFC01]. The algorithm is designed to take a variable-sized binary input less than  $2^{64}$  bits and output a 160 bit message called a "message digest."

The SHA-1 hash function is used to hash the information dictionary found in the .torrent file. The digest represents a digital signature of the file and its contents to prevent confusing different files with the same file name. According to the BitTorrent Protocol Specifications, "The peer wire's protocol consists of a handshake followed by a never-ending stream of length-prefixed messages. The handshake starts with character nineteen (decimal) followed by the string `BitTorrent protocol`" [Coh08]. The decimal nineteen followed by the string `BitTorrent protocol` is critical for identifying BitTorrent packets in this research. The next piece of information is the 20 byte SHA-1 hash of the information dictionary [Coh08]. An example of a handshake message is dissected below [Sch09].

The client sends handshakes to other peers to retrieve parts of the file:

```
<13>BitTorrent protocol<00000000000100001101C9D63211C3C570FFBA  
DD49C5649D3FB4972732D5554313737302DF39FFDC774B56A4C5352C11C>
```

Line breaks and spaces are added to aid in readability:

```
<13>BitTorrent protocol
<00 00 00 00 00 10 00 01
10 1C 9D 63 21 1C 3C 57 0F FB AD D4 9C 56 49 D3 FB 49 72 73
2D 55 54 31 37 37 30 2D F3 9F FD C7 74 B5 6A 4C 53 52 C1 1C>
```

The first piece of information in the extracted handshake is the string length of the protocol being used (0x13 in hexadecimal is 19 in decimal.) The second piece is the protocol header, the ASCII string “BitTorrent protocol” which is 19 characters in length. The third portion consists of the reserved extension bytes 00 00 00 00 00 10 00 01. This is followed by the SHA-1 hash of the information dictionary:

```
10 1C 9D 63 21 1C 3C 57 0F FB AD D4 9C 56 49 D3 FB 49 72 73
```

and finally the Peer Identification:

```
2D 55 54 31 37 37 30 2D F3 9F FD C7 74 B5 6A 4C 53 52 C1 1C
```

The BitTorrent protocol relies on the transfer of blocks of data that combine to form the file being shared. As a result, these blocks are also run through SHA-1 to ensure data integrity. The client performs a SHA-1 on each downloaded block and compares it to the value in the .torrent file to verify the data integrity [Coh08].

*2.1.2 The Voice over Internet Protocol.* The Voice over Internet Protocol (VoIP) is primarily used to make phone calls over the Internet [Sky09]. A person’s voice is digitized, placed in a packet, and sent to the receiver on the other end. The primary protocol used to setup, maintain, and tear down a VoIP call is the Session Initiation Protocol.

*2.1.2.1 The Session Initiation Protocol.* The plan and protocol for the Session Initiation Protocol (SIP) was submitted by Henning Schulzrinne of Columbia University in 1999 [Ubi08]. The protocol, approved by the Internet Engineering

Task Force (IETF) as Request for Comments (RFC) 2543, centered on establishing and controlling multiparty multimedia sessions [Ubi08]. The SIP protocol was updated in IETF RFC 3261 and defined to be an application-layer control protocol that can establish, modify, and terminate multimedia sessions such as Internet telephony calls [RFC02]. The purpose of SIP is to assist in peer location in addition to managing the connection once it is established. Applications such as interactive gaming, media on demand, and voice, video, or web conferencing utilize the SIP protocol [Ubi08]. More importantly, SIP is currently used by VoIP providers Vonage and Skype [Cis02] [Sky09].

*2.1.2.2 VoIP Technical Specifications.* Figure 2.1 is an illustration of how a VoIP call is made using SIP. For clarification, Alice is calling Bob using a Proxy Server that coordinates and routes requests between clients and servers. The call also uses a Registrar/Location Server, which is a database of all the SIP clients and SIP contact information within a network domain. Lastly, BYE and INVITE are SIP specific request messages.

- Alice's SIP client sends an INVITE request to her Proxy Server (1). Alice's Proxy Server notifies her that a call is being attempted (2). Alice's Domain Name System (DNS) server must perform a DNS lookup to determine the Internet Protocol (IP) address of Bob's domain (3,4).
- Alice's Proxy Server sends the INVITE request to Bob's Proxy Server (5). Bob's Proxy Server notifies Alice's Proxy Server that a call is being attempted (6). Bob's Proxy Server must query the Registrar/Location Server to determine Bob's exact location and if he is currently signed on (7,8). Bob's Proxy Server forwards the INVITE message to Bob's SIP client (9).
- Bob sends his ringing response back to Alice via the Proxy Servers (10,11,12). If Bob is available, his SIP client sends an OK response to Alice via the Proxy Servers (13,14,15).



- Alice receives Bob's OK and sends an acknowledgment directly back to Bob to confirm the call (16). The session is now established and data can be exchanged via the Real-time Transport Protocol [Cis10].

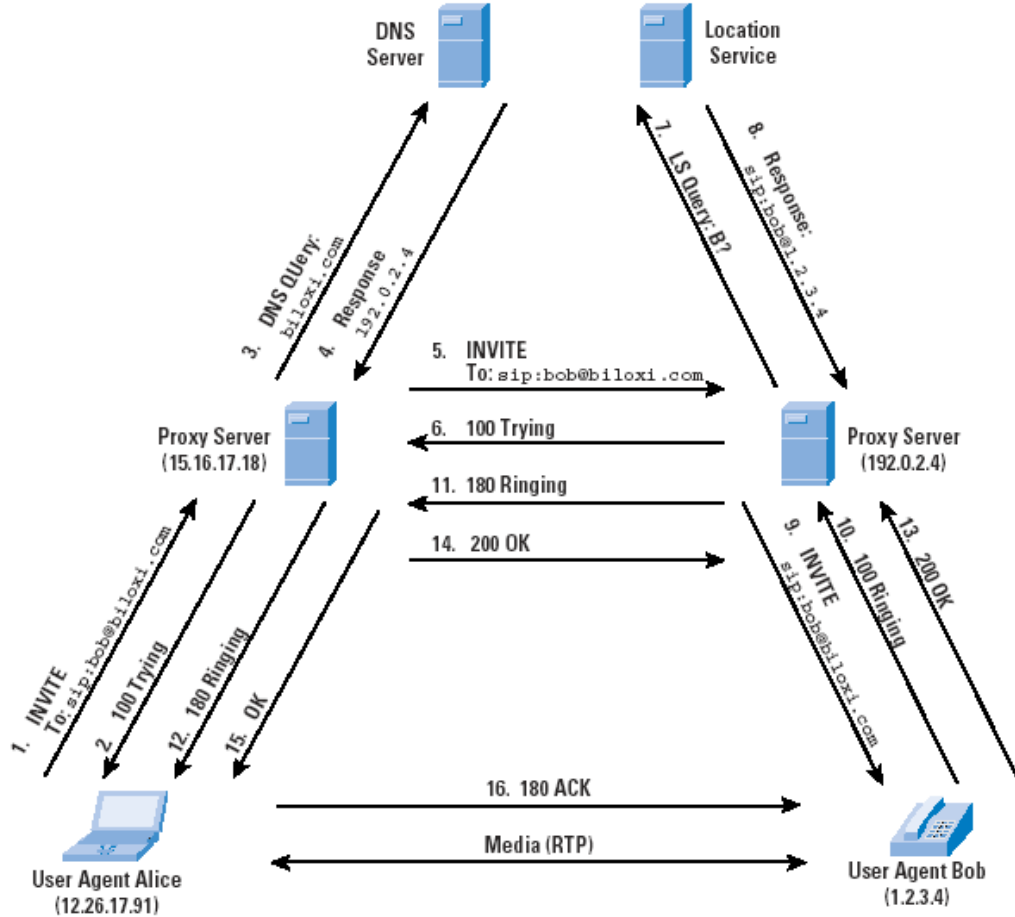


Figure 2.1: The Session Initiation Protocol Process [Cis10].

The steps of interest are at the beginning and end of the SIP-based VoIP call, particularly the session setup and tear down. During the setup of a SIP-based VoIP call, the INVITE request is used to initiate a connection from one client to another. The INVITE request contains the SIP Uniform Resource Identifier (URI). The URI is the address of the client on the network and follows the same formatting convention as an email address (`user@host`). In the SIP message, the SIP URI is concatenated to the `sip:` identifier. Examples include `sip:bob@example.com` and `sip:2001@10.1.1.1`. During the tear down of a SIP-based VoIP call, the BYE request messages are used to

terminate a SIP connection session. The BYE message also contains SIP URIs, which can identify certain users or domain addresses. For the purpose of this research, the INVITE and BYE SIP requests are examined in the SIP transaction because they contain the URI of both the sender and receiver.

*2.1.3 The Domain Name System.* The Domain Name System (DNS) is perhaps the most critical service for the Internet. DNS converts human-friendly host addresses to computer readable Internet host addresses, much like a phone book. This allows a user to remember `google.com` instead of the IP address `74.125.67.100`. Despite the massive dependency on DNS, the security and vulnerabilities of the protocol have recently come to light. A brief overview of how DNS works is followed by the current methods of abusing the protocol, specifically DNS tunneling.

*2.1.3.1 The Domain Name System Overview.* DNS is a distributed database that is indexed by domain names with the goal of decentralized administration. The domain name is part of a path in an inverted tree that constitutes the domain name space. As shown in Figure 2.2, the top of the inverted tree contains the root, with various subdomains that branch off from it [AL01].

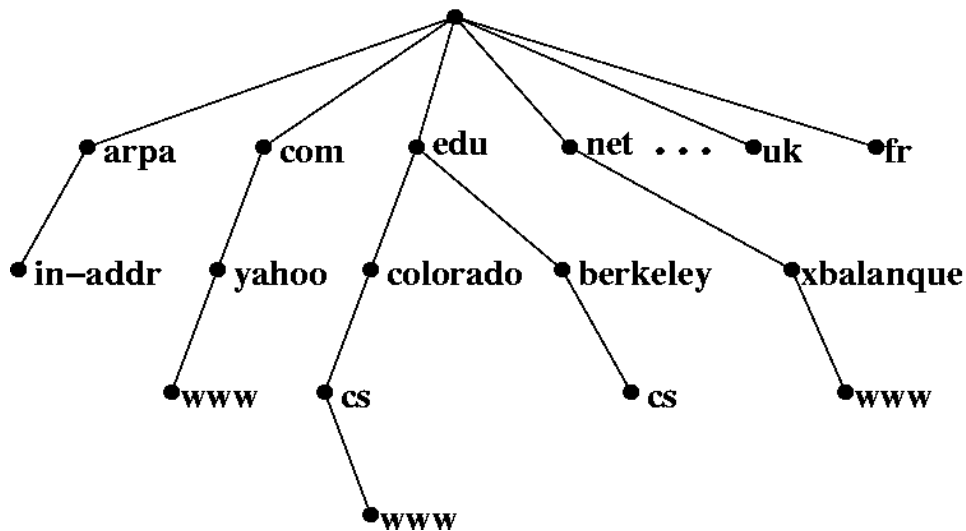


Figure 2.2: The Domain Name System Distributed Database [Moh09].

Each of the nodes, or domain names, contain a text label that is capped at 63 characters in length as opposed to the root which is a zero-length, or null, label. The full domain name of a node starts from the node and follows the path up towards root, adding each subsequent node to its name [AL01]. In the example from Figure 2.2, this would be `www.cs.colorado.edu`.

*2.1.3.2 Domain Name System and the Internet.* DNS is implemented on the Internet to create one of the most crucial infrastructure services. The Internet domain name space consists of certain top level domains. Some of these top level domains include .com, .edu, .mil, and .gov and are managed by the Internet Corporation for Assigned Names and Numbers (ICANN). With the knowledge of the top level domains, it is easier to dissect and read domain names. The decentralized administration of DNS is possible through delegation. Delegation allows domains and subdomains to be broken up for ease of management [AL01]. The personnel who run the .mil domain would rather delegate responsibility to the subdomains, such as `af.mil` and `navy.mil`, than manage each of them.

A key component of a domain name space is the name server. Name servers contain information pertaining to the domain name space, also called a zone. There are two types of name servers: secondary master and primary master. The secondary master for a zone name server polls the primary master server for zone data. The primary master name server for a zone extracts the data for the zone from a local file, called zone data files, which are also referred to as data files or database files. The zone data files contain resource records describing the hosts and delegation subdomains in the zone. These zone data files also contain entries called DNS resource records [AL01].

Each of the domains in a domain name space contains resource records that contain data associated with the domain. Some of these records include A for address record, NS for nameserver record, CNAME for canonical name record, and TXT for text record [AL01].

Another key component of the domain name space is the resolver, which is a client that accesses name servers when information is needed. An example is a web browser trying to determine the IP address of `google.com`. The resolver's three tasks are to handle querying a name server, interpreting responses, and passing the information to the requesting program [AL01].

The resolution process is important for name servers to retrieve data from the domain name space for the resolvers. Name servers perform two functions in the domain name space. The first is to resolve data within their own authoritative zones (within their own organization's network). The second function is resolve data for non-authoritative zones in the domain name space (from another organization's network). The resolution of data by name servers is accomplished either recursively or iteratively. In the recursive case, name servers pass the responsibility to more authoritative name servers to resolve data. The iterative process requires a single name server to query other name servers to try and get closer to the actual answer [AL01].

*2.1.3.3 Abusing the Internet Domain Name System.* The reliability, speed, and dependence on DNS make it a critical service for the Internet. However, the DNS protocol can be taken advantage of for nefarious purposes. One method of abusing the protocol is DNS tunneling as first suggested in a 1998 Bugtraq posting by Oskar Pearson [Pea98]. DNS tunneling is an abuse of DNS records to transfer non-DNS data in and out of a network using the DNS protocol. Non-DNS data can include files, botnet commands, and even segmented audio media [Van09]. DNS tunneling is appealing because it is a covert channel and is operating system independent [Van09]. DNS tunneling contrasts with legitimate tunnels, such as Virtual Private Network and Secure Shell, which are explained in Section 2.3.

The concept of the DNS tunnel is to use a DNS server, under control of a hacker, as an external trusted server to tunnel information out of a protected network through User Datagram Protocol (UDP) port 53. Since most protected networks permit DNS traffic to exit, the requests are granted. The data are transmitted through the tunnel

by sending data to the hacker's DNS server in the form of a query and getting data back in the form of a response [Van09]. This can be done once to communicate with a botnet or repeated thousands of times to exfiltrate files and data. The tunneled data appears as the DNS request, `[exfiltrated data].hacker.com`, with the data residing in the lowest level domain. The `[exfiltrated data]` is usually encoded in Base32 or Base64 and would look more like `0adbEnPJygrGCgvGS.hacker.com` if it was viewed using a network protocol analyzer such as Wireshark. Since `hacker.com` is under the hacker's control, the DNS server interprets the request according to the hacker's desires. The hacker's DNS server decodes the exfiltrated data and then responds with data that is tunneled back to the compromised computer in the form of a DNS response [Van09]. Figure 2.3 summarizes the process in five steps:

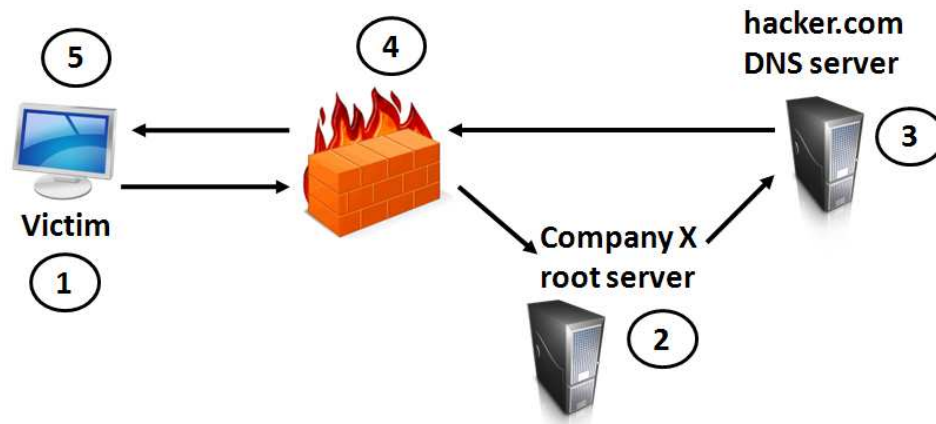


Figure 2.3: Establishing a Domain Name System Tunnel.

1. The victim's computer performs a DNS request for `[exfiltrated data].hacker.com`.
2. `[exfiltrated data].hacker.com` is not locally cached, so the victim asks the Company X root DNS server if it can resolve the request.
3. Company X's root DNS server cannot resolve the request, so it forwards it to the DNS server under the hacker's control at `hacker.com`.

4. The hacker sends back a DNS response which easily passes through a network defense appliance since DNS is assumed to be trusted.
5. The victim receives the DNS response to exfiltrate more data, connect to a botnet, etc.

The amount and type of data transferred through a DNS tunnel depends on the DNS record being used. Some of the commonly abused DNS records to tunnel data include:

- **TXT:** Text records permit free form data and can include spaces. Information stored is encoded in Base64 allowing 220 bytes of data per record. TXT records can contain any data in them as long as the length is less than 255 octets.
- **CNAME:** Canonical Name Records are alias records. They only allow the characters A through Z, digits 0-9, and the hyphen.
- **EDNS0:** The Extension Mechanism for DNS record can be greater than the 512 byte UDP DNS maximum and carry a 1280 byte default payload.
- **A and MX:** Address and Mail records, respectively. They can be used as well, but have more limitations and cannot store all types of data [Van09].

#### *2.1.3.4 The OzymanDNS Domain Name System Tunneling Application.*

OzymanDNS is a tunneling program used to tunnel all Internet traffic through DNS. It accomplishes this by encapsulating Internet data as DNS traffic and sending it through UDP port 53 instead of the traditional HTTP TCP port 80. It was developed by the DNS security guru Dan Kaminsky. This program allows users to discreetly send traffic through port 53 and the DNS protocol. Organizations and agencies rely on DNS to provide domain name resolution and lookups for the network and users. The ability for malicious users to transport possible data and traffic through a trusted and overall innocuous port is a major threat. The OzymanDNS suite of perl scripts is 32 kilobytes, making it a small and efficient exfiltration tool [Kam09].

#### *2.1.3.5 The Iodine Domain Name System Tunneling Application.*

Iodine is another program capable of tunneling Internet Protocol version 4 traffic through DNS. Iodine offers more benefits over other DNS tunnel implementations. Some of these benefits include portability between systems, a Message-Digest algorithm 5 (MD5) challenge-response for login, and the use of the NULL type to allow unencoded downstream data which allows up to a kilobyte of compressed payload data. The single program can operate as a client or server depending on the options specified by a user. Iodine supports, in decreasing bandwidth order, the use of NULL, TXT, SRV, MX, CNAME and A records [Kry09]. For this research, the malicious DNS packets used for testing are created using Iodine.

## ***2.2 Analyzing and Classifying Network Traffic***

Before the advent of Darknets and anonymizers like Tor (see Section 2.3), analyzing network traffic was relatively simple [BEPW02] [Tor09a]. The three methods for analyzing network traffic are port matching, payload analysis, and transport-level communication flow.

*2.2.1 Port Matching Analysis.* The most rudimentary, although sometimes most effective, method of classifying network traffic is done by port matching analysis. The transport level source and destination ports are extracted to reveal which ports are being used. By comparing the ports to a known list of protocols, the traffic can be classified quickly and efficiently.

Simple examples include the use of port 23 for telnet or port 80 for web servers. Ports 23 and 80 are well known ports for their respective services, although they are not bound to the ports. A protocol of interest in this research, DNS, runs over both TCP and UDP port 53. Identifying traffic on UDP port 53 will be critical in detecting DNS exfiltration attempts.

The primary problem with port matching is that some applications are not anchored to a port or port range. For example, the BitTorrent protocol is not anchored

to a single port. BitTorrent users have the option to manually assign a port number to use or allow the client program to randomly assign one. Malicious insiders could potentially run BitTorrent through port 80, a port open on most enterprise networks for web traffic [Gon05]. Only a detailed packet inspection would reveal the true nature of the traffic. Research shows up to 70% of Internet traffic is unidentifiable strictly based on port, underlining the futility in identifying traffic solely based on port matching [MW06].

*2.2.2 Payload Analysis.* The next step in classifying network traffic can be done by analyzing the payload. The payload of packets contains certain byte strings signifying the use of a certain application or protocol. Sen et al. developed an approach to identify peer-to-peer protocols based on application-level signatures. The protocols researched were Gnutella, eDonkey, DirectConnect, Kazaa, and BitTorrent. With the BitTorrent protocol, there is no signaling traffic between the client server and tracker server. Sen, et al. identified BitTorrent traffic by the distinct BitTorrent handshake message [SSW04]. The BitTorrent handshake message has the following format:

`<0x13><BitTorrent Protocol>`

The BitTorrent 20-byte signature is at a fixed location in the payload making its identification accurate [SSW04]. This makes payload analysis an attractive method for detecting BitTorrent traffic because of the unique signature. Additionally, Sen, et al. found a virtual 0% false positive and 10% false negative detection rate for identifying the peer-to-peer traffic [SSW04].

Payload analysis is effective in identifying BitTorrent payloads that have not been obfuscated. The simple technique of byte padding can render these payload-based analyzers useless, unless they are modified to search through the entire payload for the specific string. Network traffic and payload obfuscation methods are discussed in Section 2.3.



*2.2.3 Behavioral Analysis.* The last method of classifying network traffic is to examine it at the transport layer. Karagiannis, et al. developed a systematic method of identifying peer-to-peer traffic flows at the transport layer while relegating the accuracy of the previously discussed port matching and payload analysis methods [KBFC04]. Their methodology focuses on two metrics when analyzing packet headers to detect peer-to-peer traffic flows. The first metric is to observe source-destination IP pairs that are using both TCP and UDP transfers, a common mark of peer-to-peer protocols. However, other applications also use TCP/UDP pairs, such as DNS, Network Basic Input/Output System (NetBIOS), Internet Relay Chat (IRC), and gaming applications, so those application layer protocols are ignored [KBFC04].

The second metric is observing connection characteristics of {IP, port} pairs. When a host joins a peer-to-peer network, it consults its starting host cache for the IP address of other peers or servers. After a connection is established between the host and another peer, the host advertises its IP address and port number to receive connections. It is essentially the host's identification in the peer-to-peer network. When twenty different peers decide to connect to the host, the traffic will reveal twenty distinct IP addresses with twenty distinct source ports all connected to the host. The equality of distinct IP addresses and ports (e.g., 20 distinct IP addresses with 20 unique ports) signifies a probable peer-to-peer connection [KBFC04].

Behavioral Analysis is effective in quickly identifying traffic based on the 5-tuple {source IP, source port, destination IP, destination port, transport layer protocol} across a network backbone. It can also be used to identify new peer-to-peer applications or protocols that have been modified. There are several drawbacks with this heuristic for detecting peer-to-peer traffic. First, the method cannot detect the specific peer-to-peer protocol or the payload being transferred. Furthermore, the BitTorrent protocol is not one of the six peer-to-peer protocols analyzed that relies on TCP/UDP pairs. Secondly, the 95% peer-to-peer flow detection rate, coupled with the 8% to 12% false positive rate, makes it an effective, but not guaranteed, method of detecting peer-to-peer traffic [KBFC04].

### 2.3 *Network Traffic and Data Obfuscation Methods*

In certain cases it is important to encrypt data and traffic for security reasons, such as online credit card transactions or when sending confidential emails. It is also important in some cases for journalists, whistleblowers, and citizens of repressed regimes to retain anonymity. However, the same methods of encrypting, obfuscating, and anonymizing data can be used for illicit purposes. These methods include byte padding, Ron’s Code 4 (RC4) encryption, Virtual Private Network (VPN) tunnels, and Secure Shell (SSH) tunnels, darknets, and the Tor network.

*2.3.1 Byte Padding.* Byte padding is the most primitive obfuscation method used to hide payloads in network traffic. When byte padding is used, a series of random characters is prepended to the payload to trick elementary packet analyzers (see Section 2.2.2). The packet analyzer will identify the payload as encrypted or unknown since it does not match any known payload signatures.

Although byte padding is a cheap and easy method of obfuscating data, it suffers two major weaknesses. The first is that the payload is still readable within the packet, it is simply in a different location. Smart analyzers can sequentially search for the byte string in the payload, but this takes more time. The second problem is that only the payload is obfuscated and not the entire conversation. This allows network flow-based algorithms to identify the network protocol being used.

*2.3.2 Ron’s Code 4.* The Ron’s Code 4 (RC4) algorithm was invented by Ronald Rivest from RSA Security in 1987 [Riv09] [Sta06]. RC4 is a variable key-size stream cipher that relies on single byte operations [RSA09]. The cipher performs quickly in software and has been implemented in Secure Socket Layer protocol communication, Lotus Notes, Oracle Secure SQL, and in the Wired Equivalent Privacy security for IEEE 802.11 [Tec09]. Another use of the RC4 cipher is in payload obfuscation of BitTorrent packets. The cipher can perform faster than other symmetric

stream ciphers as seen in Table 2.1. The table categorizes different ciphers, the key length, and the speed of the encryption cipher in Mbps.

Table 2.1: Speed Comparisons of Symmetric Ciphers on a Pentium II [Sta06].

Cipher	Key Length	Speed (Mbps)
DES	56	9
3DES	168	3
RC2	variable	0.9
RC4	variable	45

*2.3.3 Virtual Private Network and Secure Shell Tunnels.* One method of obfuscating network data and the traffic is to use a tunnel. Tunnels allow one protocol to be transferred over another protocol. Common tunneling applications include VPN and SSH. Gebski et al. assert the difficulty in identifying the underlying protocols because the entire packet is scrambled and encrypted, as are any useful fields in the TCP/IP header [GPW06].

In the VPN and SSH obfuscation methods, an encrypted tunnel is established between the downloader and uploader. Network data and traffic are transmitted only after the tunnel has been established. The network data and traffic are encapsulated with an SSH or VPN header, thus encrypting the entire network conversation instead of just the payload. Analyzers of the network traffic will only be able to identify the source and destination IP addresses, approximate packet size, and timing of the traffic [GPW06]. Despite the limited information leaked by VPN and SSH connections, Gebski et al. correctly identified encapsulated BitTorrent traffic 90.5% of the time using bipartite graphs of outgoing-incoming node pairs [GPW06]. This discovery was supported when Wright et al. were able to accurately track the flows of encrypted tunnels carrying a single application protocol [WMM06]. It should be noted that both of these methods of inferring the underlying traffic are still unable to conclude what the packets contain.

*2.3.4 Darknets.* An even more clandestine form of file sharing and communication is rising in the form of Darknets. Biddle et al. of Microsoft first addressed the rise of content distribution and peer-to-peer networks in their 2002 paper entitled “The Darknet and the Future of Content Distribution.” They concluded that Darknet-based peer-to-peer file sharing technologies were growing in convenience, bandwidth, and efficiency and would not likely encounter technical impediments [BEPW02].

The definition of a Darknet has evolved from any public peer-to-peer network, such as BitTorrent, Usenet, and Gnutella, to any network that is friend-to-friend oriented. These Darknets, as opposed to the public Lightnets, are based on a “members only” camaraderie and trust between members. Darknets are, in the truest sense, nearly impossible to find. However, there is software available to the public to establish Darknets [Fil07]. Two of the most popular Darknet software applications are Freenet and WASTE. The availability of public software to join and establish private Darknets makes it difficult to detect illicit file transfers and possible private VoIP connections.

*2.3.4.1 Freenet.* Freenet is a software application allowing users to publish and retrieve information without the fear of being censored [Fre09]. Freenet can be described as an “Internet within an Internet” that relies on encrypted communication between other nodes. Users contribute to the Freenet project by providing bandwidth for routing and a piece of their hard drive, called the data store, to hold encrypted data. The user is oblivious to the content being stored in the data store, thus making it difficult for prosecution of possession of illegal or copyrighted material. The data is automatically added and deleted based on the popularity of certain content and the needs of Freenet [Fre09].

*2.3.4.2 WASTE.* WASTE is a software application and protocol that caters to smaller groups of 10-50 nodes. It provides an anonymous, secure, and encrypted collaboration tool to share ideas and data [WAS09]. WASTE implements a decentralized distributed architecture for nodes to create a partial mesh network.

Security features include link-level encryption using Blowfish and RSA public keys for authentication. This application allows trusted users to securely trade possibly illicit or illegal files with each other [WAS09].

*2.3.5 The Onion Router Network.* The last method of obfuscating and hiding network traffic data is The Onion Router (Tor) network. Tor is the most popular and “good intentioned” anonymizer allowing users to maintain privacy and security on the public Internet through numerous layers. The benefits of Tor include security and privacy by using a distributed network of relays to bounce traffic. It is supposed to prevent monitoring and the revealing of your physical location [Tor09a]. These indirect and random data pathways make it difficult for sophisticated traffic analysis to take place.

Tor could technically be considered a Darknet, but the intentions and goals of the project seem to separate it from the negative connotations associated with Darknets. Some government entities use Tor as well. The United States Navy uses Tor for open source intelligence gathering and law enforcement uses Tor to anonymously survey web sites [Tor09b]. The technical aspects of Tor are similar to Freenet and WASTE, in which users can voluntarily route traffic throughout the network.

Tor differs from VPNs and other encrypted tunnels in that it is not susceptible to timing and communication analysis. Despite the best intentions of the Tor network, illicit file sharers can still use it to transfer illegal material.

## ***2.4 Current Methods for Detecting Illicit Traffic***

There are many methods currently available to detect illicit traffic. This sections covers the simpler, software-based solutions such as using Snort rules and progresses towards the more intelligent solutions that utilize artificial intelligence.

*2.4.1 Wireshark.* Wireshark is one of the most popular network protocol analyzers. It is the standard for analyzing traffic and simple network troubleshooting.

It runs as a software application on a system and requires a network card that can be set to promiscuous mode. Wireshark will display all of the incoming and outgoing packets on an interface, but allows the user to filter the results by protocol, IP address, or port number, to name a few. The advantages of Wireshark are that it is free, reliable, and easy to use on a small scale. The disadvantages of Wireshark include operating at the application layer and the inability to perform complex traffic analysis [Wir09].

*2.4.2 Snort.* Snort is an open source intrusion detection and prevention system designed to be implemented in software. It is a rule-based application that can perform real-time traffic analysis and packet logging on IP networks. Snort is capable of protocol analysis, content searching and matching, and attack detection by relying on a flexible rule set used to describe the handling of certain traffic. Snort has three modes of operation: packet sniffer, packet logger, or a complete Intrusion Prevention System [Sno10]. Snort is a powerful and highly-regarded Intrusion Prevention System for providing network security. However, Snort has two shortfalls. First, Snort must be installed and run on a dedicated and powerful computer because of the processor-intensive rules. Large rule sets can deteriorate the performance of Snort if it is processing all inbound and outbound traffic. This scenario could lead to possible missed critical BitTorrent, SIP, or DNS packets. Secondly, since Snort and the Snort rules are open source, the code can be analyzed to determine how to avoid detection [Sno10].

*2.4.3 Hi-Performance Protocol Identification Engine.* The Hi-Performance Protocol Identification Engine (HiPPiE) is another software-based protocol analyzer. It differs from Wireshark in that it attempts to analyze traffic and protocols heuristically. Some of the more impressive features include Session Prediction Support and Tunneled Protocol Tracking. The Session Prediction Support has the ability to predict upcoming protocol sessions. The Tunneled Protocol Tracking feature identifies the internal protocol being used with a tunneling protocol [HiP09b].

The HiPPIE advertises three main functionalities [HiP09a]. The main functionalities include:

1. Passive traffic analyzer: HiPPIE must be installed in the kernel of a Linux system and configured to push traffic to a single-sourced bridge interface.
2. Inline Protocol/Packet Filter: This method entails establishing a Linux system with HiPPIE as either an in-line bridge or routing device that forces traffic through using integrated Netfilter or IPTables to filter traffic. It also allows network administrators to tag or limit certain types of traffic based on HiPPIE's recognition capabilities.
3. Plug-in to a third party system: This option, although not completed, allows administrators to pass traffic from a traffic sniffing application, such as tcpdump, to be analyzed by HiPPIE.

The downside of the HiPPIE system is that it does not perform payload inspection, only protocol analysis [HiP09b]. Payload inspection consists of examining the contents of a packet. Protocol Analysis only inspects the headers to determine what protocol is being transmitted.

*2.4.4 BitTorrent Monitoring System.* Another method of detecting and tracking illicit files is the BitTorrent Monitoring (BTM) system, created by Chow, et al. The BTM is an automatic, rule-based software application to monitor, record, and analyze BitTorrent traffic [CCM<sup>+</sup>07]. Figure 2.4 illustrates how the BTM system works. The BTM is divided into two different modules, the Torrent Searcher and Torrent Analyzer.

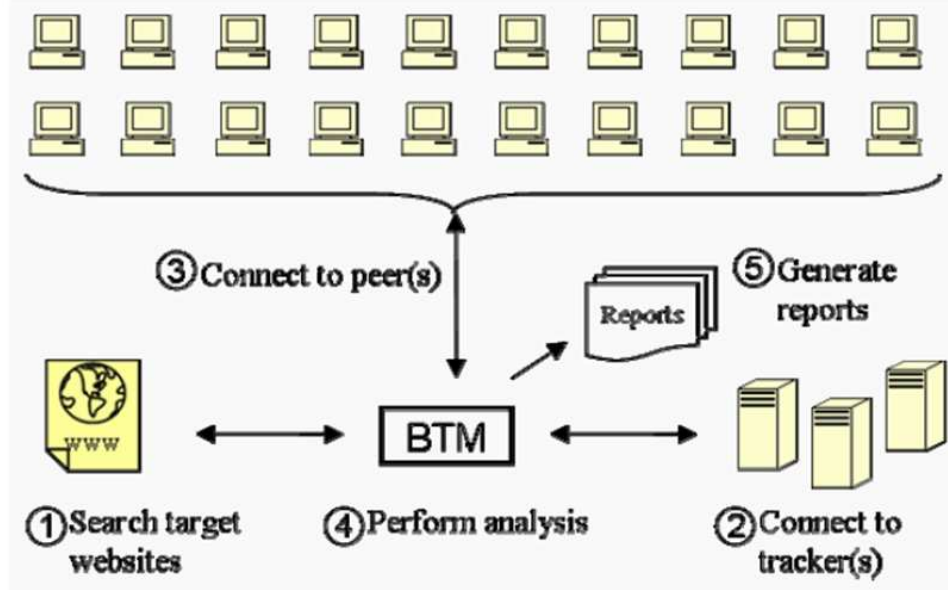


Figure 2.4: The BitTorrent Monitoring System Process [CCM<sup>+</sup>07].

The Torrent Searcher is a passive reconnaissance function to collect torrents of interest. The BTM commences by searching public forums and web sites for torrent files and exploring the various hyperlinks. This depth-first search continues until a predefined level has been reached. Each torrent file and webpage containing predefined keywords of interest are downloaded and archived to the local investigator's computer [CCM<sup>+</sup>07].

The Torrent Analyzer is the interactive portion of the BTM since it communicates with trackers to retrieve the list of peers sharing the file. Responses from the trackers and peers are recorded by the BTM for future analysis. The BTM has a real-time attributed-based rule engine to flag specific tracker or peer information [CCM<sup>+</sup>07].

The fundamentals of the BTM system are sound, but there are several concerns to consider. First, the scope of the system is limited due to the immense volume of torrent files on the Internet. Two of the larger tracker sites, [piratebay.org](http://piratebay.org) and [isohunt.com](http://isohunt.com), contain approximately 1.8 million and 1.7 million torrents, respectively [Bay09] [ISO09]. Secondly, the fluctuating list of peers associated with a torrent



changes by the minute, making the BTM system less than ideal for associating specific IPs with a file [CCM<sup>+</sup>07].

*2.4.5 Entropy-Based Malicious DNS detection.* As mentioned earlier, the DNS protocol can be abused to exfiltrate data or be used as a command and control channel for botnets. Typically, DNS traffic is minimal between clients (DNS resolvers) and servers (DNS servers). Romana et al. performed an entropy study of external DNS query traffic to the university network’s top domain server. Any peak in the entropy was assumed to be associated with spam botnet activity [RKSM08]. Figure 2.5 illustrates the entropy changes in source IP addresses and the DNS query contents-based parameters.

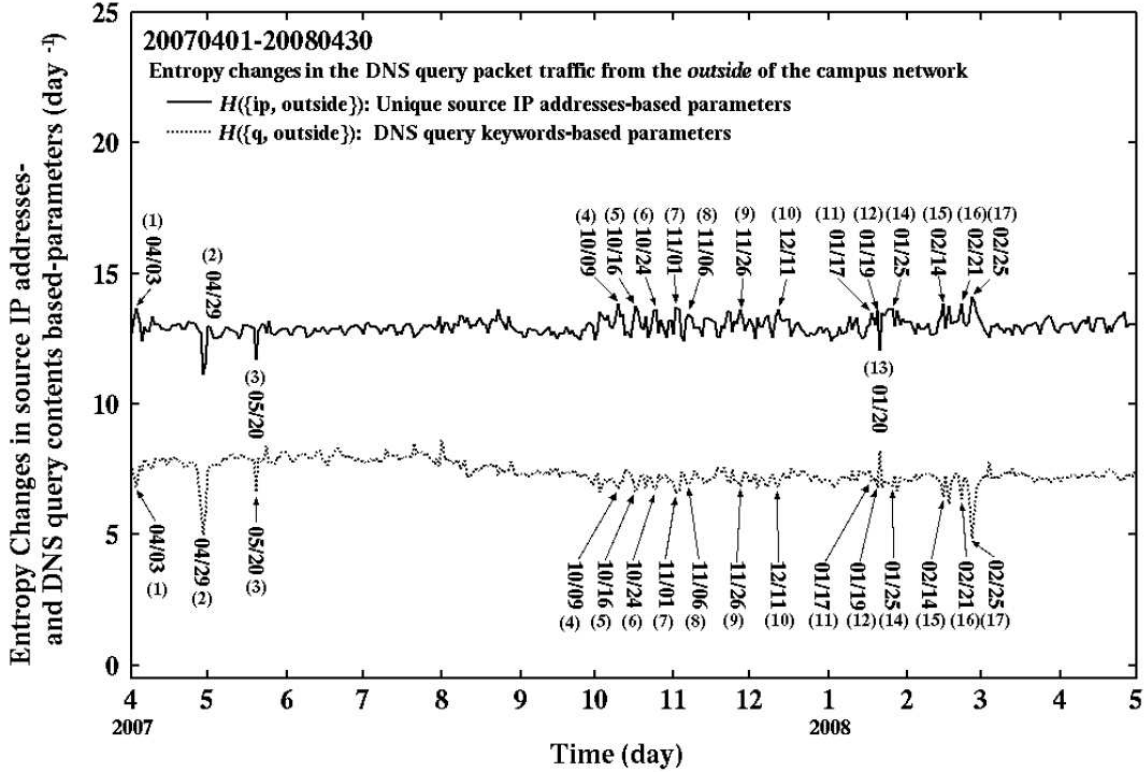


Figure 2.5: Entropy Changes In the Domain Name System External Query Traffic [RKSM08].

The drastic changes in entropy, denoted as the spikes, were hypothesized to be botnets used to communicate, exfiltrate data, or perform other malicious mis-

sions. Botnet infection was verified on the computers after performing forensic analysis [RKSM08].

*2.4.6 Cross Entropy-Based Malicious DNS detection.* Karasaridis et al. developed a DNS Tunneling Attack Detector (TUNAD) to detect suspicious DNS packet size anomalies in real-time [KMHH06]. Outlying packet sizes are usually indicative of malicious tunneling over DNS. Their approach is as follows:

1. Separate DNS packets into three types and calculate the frequency of non-conforming UDP DNS packet sizes:
  - Requests: source port, sport>1023 and destination port, dport=53. The size cannot exceed 300 bytes.
  - Response: source port, sport=53 and destination port, dport>1023. These are normally less than 512 bytes.
  - Unknown: Response or request certainty is unknown, with sport=53 and dport=53. These are normally less than 512 bytes.
2. Measure the exact packet size using single packet flow records.
3. Calculate hourly packet size histograms for each circuit and packet type.
4. Use a Cross Entropy-based anomaly detector on the packet size histograms.

The algorithm then computes the Cross Entropy, Self Entropy, and Relative Entropy to detect anomalies. The algorithm was successful in detecting a change in Relative Entropy of packet sizes on September 30, 2003 before reports surfaced about the Sinit Trojan that used port 53. This method of calculating changes from a relative baseline is important in detecting suspicious DNS traffic [KMHH06].

*2.4.7 Detecting DNS Tunnels Using Artificial Neural Networks.* In 2009, Jhind presented research on detecting DNS tunnels using artificial intelligence called dnsTTrap [jhi10]. The algorithm relies on Artificial Neural Networks. The algorithm, also called supervised learning, works as follows:

1. Receive inputs (number of packets to domain, average length of packets to domain, average number of distinct characters in the lowest level domain)

2. Give them values (assign weights)
3. Adapt decisions until inputs match training data (set thresholds)

The goal was to examine the entropy of the data contained in the lowest level domain. The reasoning is that if data is being exfiltrated by the lowest level domain, the content of each lowest level domain will be entropic. Each lowest level domain is assigned a numerical value, allowing comparison between other lowest level domains. For example, the domains `mail.example.com` and `mail2.example.com` will have minimal entropy between them. However, the domains, `4ryf76df.hacker.com` and `73bfdd7r.hacker.com` will have greater entropy and can be classified as a possible DNS tunnel. The last step is to train the neural net using data controlled by the user. False negatives are added to the training list and the system is retrained [jhi10].

The system managed to detect DNS tunnels created by the DNS tunneling applications Iodine, OzymanDNS, and tcp2dns. However, the system lacks real-time detection since it only works against previously captured tcpdump files. In addition, the system only analyzes the lowest level domain, instead of the entire domain. DNS tunneling applications can be modified to transfer data at different levels of the domain, such as `exfiltrated_data.mail.example.com` to `mail.exfiltrated_data.example.com` [jhi10].

## ***2.5 The Substitute Database Manager Hashing Function***

A new feature of the TRAPP-2 system is the implementation of a hashing function used in the Substitute Database Manager (sdbm) library. The hashing function converts arbitrary-length strings into eight-byte hashes. The arbitrary-length strings in this case are SIP URIs and DNS domains. The sdbm hashing function is selected over more proven hashing functions such as SHA-1 and MD5 because it is quick and easy to implement. Further justification of the feature can be found in Section 3.2.2.

In December 1990, Ozan Yigit released the sdbm library into the public domain as an alternative to the original Database Manager (dbm) database engine, and

subsequent New Database Manager (ndbm) database engine, developed by AT&T in 1979 [Yig10b] [SY91]. The sdbm library is a clone of the ndbm library and parallels the functionality. However, the sdbm library relies on the simple hashing algorithm found below, implemented in the C programming language [Yig10a].

```
static unsigned long sdbm(unsigned char *str){
    unsigned long hash = 0;
    int c;
    while (c = *str++)
        hash = c + (hash << 6) + (hash << 16) - hash;
    return hash;
}
```

According to the creator, Ozan Yigit, sdbm “was found to do well in scrambling bits, causing better distribution of the keys and fewer splits. It also happens to be a good general hashing function with good distribution” [Yig10a]. The hashing function’s speed (See Appendix B), eight-byte hashes, and easy software implementation made it an ideal hashing function for the TRAPP-2 system. While using the hashing function, one drawback occasionally noted is the minimal avalanche effect in which changing a DNS domain by one bit (e.g., from 123.com to 124.com) changes the hash by one bit. Another possible drawback is the number of collisions between hashes. This, however, is not investigated since the assumption is that an administrator will review the packets and their hashed domains.

## ***2.6 The Tracking and Analysis for Peer-to-Peer System***

A FPGA-based packet analyzer was developed in 2008 to detect peer-to-peer protocols traversing a network. The TRAPP system was built specifically to detect BitTorrent and VoIP traffic. The TRAPP system was created as an alternative to current illegal file detection techniques such as software packet sniffers and the BitTorrent Monitoring System. A discussion of these various techniques can be found in Section 2.4. The current TRAPP system is built on a Xilinx Virtex-II Pro FPGA board. The TRAPP system, capabilities, and limitations are expanded to better understand the state of the system [Sch09].

*2.6.1 Capabilities.* The TRAPP system is designed to operate at the gateway between the Internet and a government local area network (LAN). It is not placed in-line with traffic entering or exiting the local network, so if the TRAPP system fails, the network will still remain viable. Instead, it is placed on the Switched Port Analyzer (SPAN) port of a switch. The switch is configured to send packets to the correct destination in addition to the SPAN port. This makes the TRAPP system virtually invisible and undetectable to both normal and malicious users. The TRAPP system extracts the BitTorrent file hash or SIP URI and compares it against a list of known contraband file hashes or SIP identifiers. The detection of the contraband files and SIP identifiers is done in real-time. Figure 2.6 is a flowchart overview of how the system works [Sch09]. The TRAPP system analyzes every packet flowing through the network switch, looking for a BitTorrent or SIP signature. If the packet has a BitTorrent or SIP signature, the hash or SIP URI are extracted, respectively. A binary search is performed on the extracted hash against a blacklist of BitTorrent hashes or SIP URIs. If a match is found, the packet is logged, else it is dropped [Sch09].

*2.6.2 Limitations.* There are several limitations with the TRAPP system. These limitations include the hardware, contraband file list size, SIP URI extraction, and lack of malicious DNS traffic detection [Sch09].

The first limitation of the TRAPP system is the hardware. The hardware components of interest on the Xilinx Virtex-II Pro FPGA board are the 100 megabit Ethernet card and 300 MHz processor [Xil08], which are suitable for smaller LANs with less traffic to compare against the contraband list. In reality, the size of government networks, traffic, and bandwidth requirements justify faster hardware.

Another drawback of the TRAPP system is the size of the contraband list. The TRAPP system relies on 64 KB of memory to store the blacklist of BitTorrent file hashes and SIP identifiers. The size of the list is limited to 1000 entries [Sch09]. This size is appropriate for a first iteration proof-of-concept system, but in reality, the list size needs to be much larger.

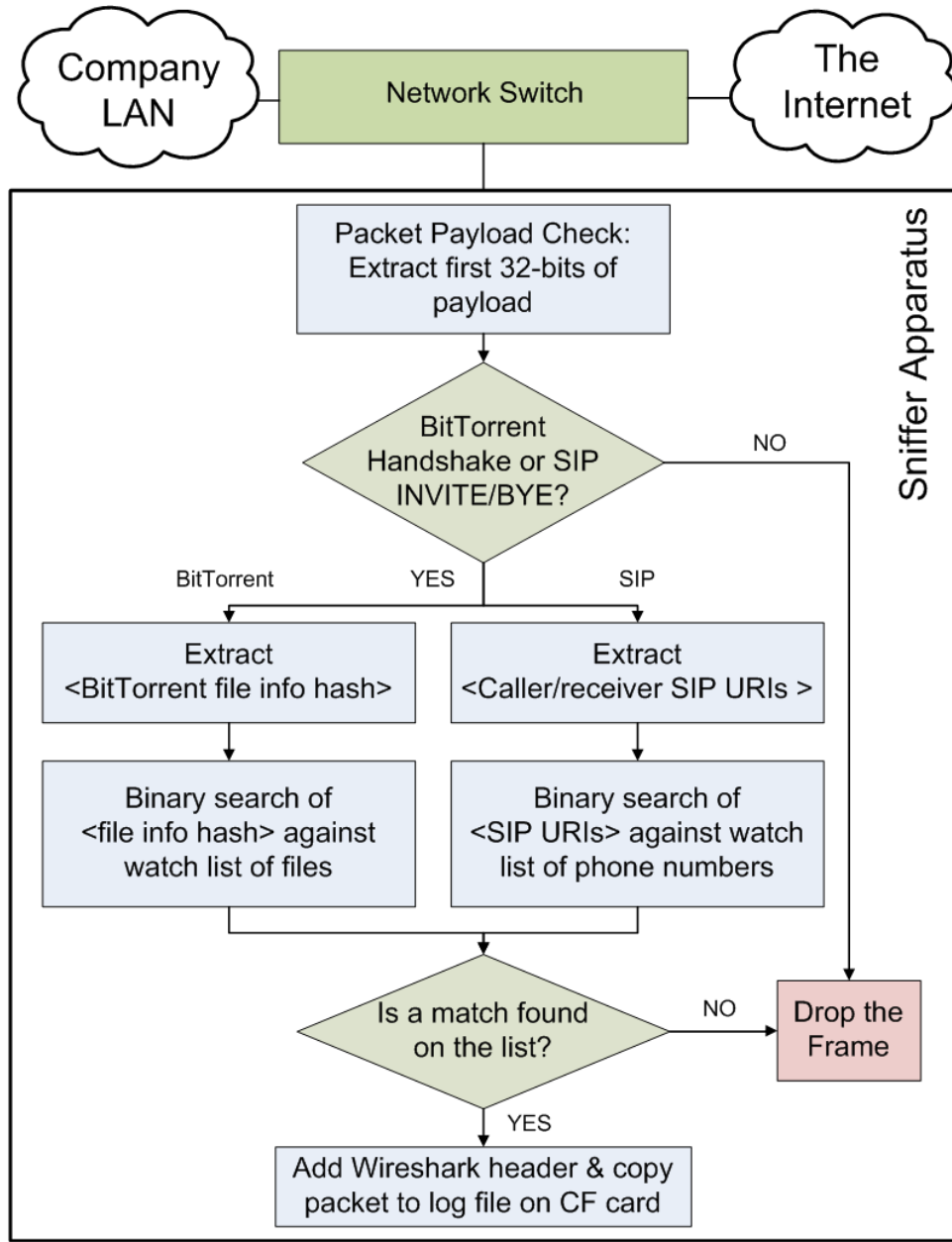


Figure 2.6: TRacking and Analysis for Peer-to-Peer System Flowchart [Sch09].

The third limitation is how TRAPP deals with processing SIP packets. The TRAPP system only extracts the first 12 bytes of a SIP URI. For example, if the SIP URI is 2001@10.1.1.50, the TRAPP system extracts 2001@10.1.1. and compares it against the list of interest [Sch09]. Although feasible, this logic is not realistic. This limitation is elaborated on in Section 3.2.2.

Lastly, the TRAPP system is unable to detect illicit DNS traffic. Hackers and malicious users abuse the DNS protocol to transfer data and information, in addition to communicating with botnets [Sch09].

## ***2.7 Summary***

This chapter discusses the illicit traffic and protocols of interest for this research, specifically BitTorrent, VoIP, and DNS. The traditional methods of classifying network traffic such as port matching are examined. This is followed by exploring the current methods of obfuscating and encrypting network data and traffic. The current methods of identifying and detecting these types of traffic, both clear and encrypted, are also summarized. The details of the sdbm hashing function are also expanded. Finally, a review of the TRAPP system's capabilities and limitations are detailed.

### III. Methodology

This chapter explains the methods used to evaluate the performance of the TRAPP-2 system. The two metrics measured are packet processing time and the probability of packet intercept. The first section details the Goals and Hypotheses. Section 3.2 outlines the Approach, and Section 3.3 outlines the System Boundaries. The System Workloads are defined in Section 3.5, followed by Performance Metrics in Section 3.6, System Parameters in Section 3.7, and Factors in Section 3.8. The last three sections include the Evaluation Technique in Section 3.9, the Experimental Design in Section 3.10, and the Summary in Section 3.11.

#### 3.1 *Goals and Hypotheses*

The objective of this research is to test and evaluate the performance of the TRAPP-2 system that detects packets of interest traversing a gigabit Ethernet network. The packets of interest include BitTorrent handshake packets with file hashes of interest, SIP Uniform Resource Identifiers (URI) of interest, and suspicious DNS traffic. The TRAPP-2 system detects these transmissions, classifies the traffic, extracts the payload (and sdbm hashes it for SIP/DNS domains), compares the hash against a list, and records the transmission information.

The goals of this research are to:

1. Determine the packet processing times for packets of interest.
2. Determine the probability of packet intercept under a flood of packets of interest.
3. Determine the probability of packet intercept under various network utilizations.
4. Determine how increasing the hash list size affects the packet processing time.



The hypotheses of this research are:

1. The TRAPP-2 system can process every type of packet under 35,000 CPU cycles.
2. The TRAPP-2 system can detect over 50% of packets of interest flooded into the system.
3. The TRAPP-2 system can detect and process BitTorrent and DNS packets with at least a 90% probability of packet intercept under a 90% network utilization. Furthermore, it is hypothesized that the TRAPP-2 system can detect and process SIP INVITE and SIP BYE packets with at least a 19% probability of packet intercept under a 90% network utilization.
4. The TRAPP-2 system's mean packet processing time will increase by no more than 50 CPU cycles each time the hash list size is doubled.

Four experiments are conducted to determine if the TRAPP-2 system meets the goals and hypotheses. Figure 3.1 summarizes the experiments, metrics, and goals used to evaluate the performance of the TRAPP-2 system.

Experiment	Metric	Goal
1	Packet Processing Time	Determine the packet processing times for packets of interest
2	Probability of Packet Intercept	Determine the probability of packet intercept under a flood of packets of interest
3	Probability of Packet Intercept	Determine the probability of packet intercept under various network utilizations
4	Packet Processing Time	Determine how increasing the hash list size affects the packet processing time

Figure 3.1: Summary of Experiments for the TRacking and Analysis for Peer-to-Peer 2 System.

## 3.2 Approach

The TRAPP-2 system is developed on the Xilinx ML510 FPGA. The reason for developing the TRAPP-2 system on an FPGA board is similar to the original TRAPP system, henceforth referred to as “TRAPP-1” in this research. The system’s simplicity and speed is maximized by allowing the software application to directly access the Ethernet controller buffers [Sch09]. In addition, hardware components can easily be added with minimum overhead. Some elements and functions from the TRAPP-1 system are used for the TRAPP-2 system, but a majority of the code is rewritten to function with the updated FPGA hardware and research goals. Although both systems work similarly, major hardware and software changes are required to achieve proper functionality in the TRAPP-2 system. A review of the TRAPP-1 system can be found in Section 2.6.

*3.2.1 Hardware Modifications.* The major hardware modification between the TRAPP-1 and TRAPP-2 systems is the Ethernet controller. The TRAPP-1 system relies on the EthernetLite core peripheral, which has an upper limit of 100 Mbps. For the TRAPP-2 system, a Trimode Ethernet Media Access Controller is used to receive Ethernet frames at 1000 Mbps. An accompanying First-In-First-Out 32,768-byte buffer stores Ethernet frames until they can be processed. As a result, the TRAPP-2 system is not strictly linked to a clock like the TRAPP-1 system.

The second hardware modification is the memory location of the hash list and log file. The hash list, separate for each of the three protocols, contains a sorted list of hashes used to determine if a BitTorrent, SIP, or DNS packet hash is of interest. The log file contains all of the packets of interest detected by the TRAPP-2 system. The TRAPP-1 system relies on two sets of 64 KB Block Random Access Memory (BRAM) to separately store the hash list and log file. The maximum amount of BRAM available on the TRAPP-2 system’s FPGA is 128 KB per block. This severely limits the maximum hash list size, which is explored in Experiment 4. As a result, the BRAM architecture is abandoned in favor of a 512 MB Synchronous Dynamic

Random Access Memory (SDRAM) scheme for the TRAPP-2 system. SDRAM is used to store the hash list and log file together. Pilot tests reveal an average increase of 777 CPU cycles in packet processing time for the SDRAM scheme. However, the 4096-fold gain in physical memory address space at the cost of 777 CPU cycles is acceptable. This memory configuration is also more realistic for future configurations that will rely on larger hash lists. The pilot test data for memory access times can be found in Appendix B. Hardware construction details for TRAPP-2 can be found in Appendix C.

*3.2.2 Software Modifications.* The first software modification adds code to detect the DNS protocol. Pilot tests reveal that the DNS detection logic requires an average of 23 CPU cycles. The DNS detection logic results in 1.37% (23/1672) of the total packet processing time for the packet with the smallest packet processing time (DNS-OFF-SMALL, which is explained in Section 3.5). The pilot test data for the DNS packet detection logic can be found in Appendix B.

The second software modification involves the processing of SIP and DNS packets. The TRAPP-1 system only extracts the first 12 bytes of a SIP URI. For example, if the SIP URI is 2001@10.1.1.50, the TRAPP-1 system extracts 2001@10.1.1. and compares it against the list of interest. Although feasible, this logic is not realistic because SIP usernames, 2001 in this case, can easily be changed. The TRAPP-2 system does not extract the SIP username, 2001, in the example. The second problem is in how the TRAPP-1 system addresses SIP URI domains, or everything after the @ symbol. The TRAPP-1 system algorithm assumes the domain is only seven characters long. As an improvement, the TRAPP-2 system extracts the entire domain and is not limited by the domain length.

The final software modification involves hashing SIP and DNS domains. For the hash lists of interest, a uniform hash length is required for proper binary searching of the hash list. The variable-length domains of both SIP and DNS do not allow for a uniform hash list. As a result, the sdhm hash is implemented to convert the

variable length SIP and DNS domains into a four-byte hash. More details about the sdbm hash can be found in Section 2.5. Pilot tests reveal that an average of 86 CPU cycles are required to sdbm hash a six character domain and 1,195 CPU cycles are required to sdbm hash a 212 character domain name. This 86 - 1,195 CPU cycle increase in packet processing time is acceptable to create uniform hash identifiers for the variable-length SIP and DNS domains. The pilot test data for the sdbm hashing times can be found in Appendix B.

*3.2.3 Algorithm.* Figure 3.2 illustrates the TRAPP-2 algorithm which includes the following steps:

1. Detect packet
2. Determine if BitTorrent, SIP, or DNS packet
3. If BitTorrent/SIP/DNS packet, extract the payload; else, discard the packet
4. If SIP or DNS, sdbm hash the domains
5. Compare the hash against the hash list
6. If a match is found (BitTorrent or SIP), log the packet; else, drop the packet
7. If a match is *not* found (DNS), log the packet; else, drop the packet

For BitTorrent packets, the system detects a BitTorrent handshake packet, extracts the first four bytes of the 20-byte SHA-1 hash, compares the hash against a blacklist containing the first four bytes of suspicious hashes, and logs it if the hash is on the blacklist. A BitTorrent packet is defined as a TCP packet with the first four bytes of the payload being 0x13426974 (“<13>Bit”).

For SIP packets, the system detects a SIP INVITE or BYE packet, extracts the entire domain from both the **To:** and **From:** portion of the SIP URI, sdbm hashes both the **To:** and **From:** domains to create two unique hashes, compares the hashes against a blacklist containing the suspicious hashes (each four bytes in length), and logs it if either the **To:** or **From:** hashes are on the blacklist. A SIP INVITE packet is defined

as a UDP packet with the first four bytes of the UDP payload being “INVI”. A SIP BYE packet is defined as a UDP packet with the first four bytes of the UDP payload being “BYE ”. It is possible for a SIP packet to have the To: and From: domains be the same. This occurs if both sender and receiver are communicating through the same SIP proxy server. Refer to Section 2.1.2 for more information about the SIP packet.

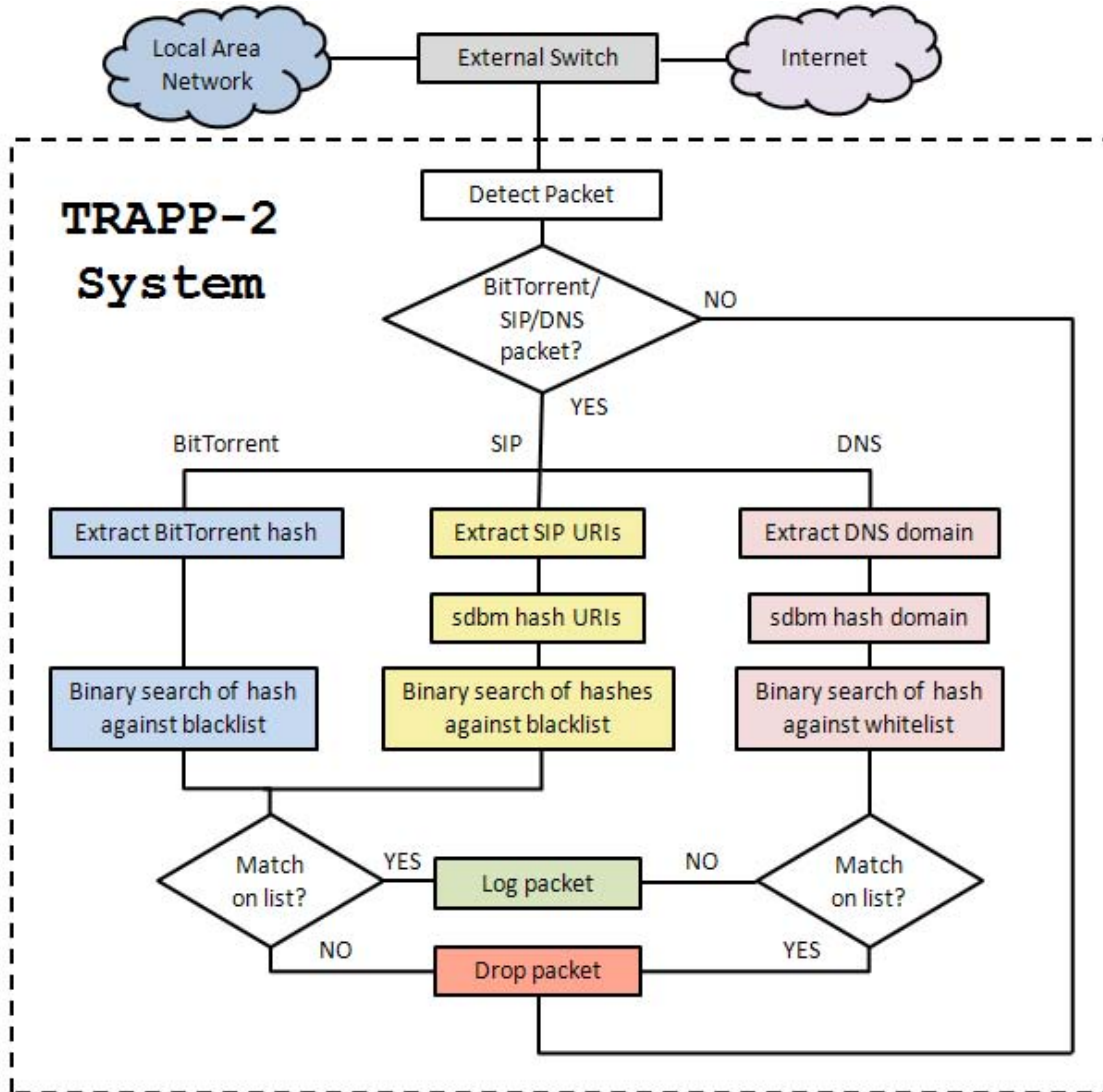


Figure 3.2: Packet Data Flow in the TRacking and Analysis for Peer-to-Peer 2 System.

For DNS packets, the system detects a DNS request, extracts the entire domain, sdbm hashes the domain to create a four-byte unique hash, compares the hash against a whitelist of approved domain hashes, and logs it if it is not on the DNS whitelist. A DNS request is defined as a UDP packet with a destination port of 53. DNS zone transfers, performed over TCP port 53, are not included because they are not capable of exfiltrating data.

### 3.3 System Boundaries

The System Under Test (SUT) for this research is the TRAPP-2 system. The SUT block diagram is illustrated in Figure 3.3. The SUT components include: TRAPP-2 software, FPGA and board, PowerPC Processor, System Timer, Ethernet Controller, two 512 MB SDRAM modules, and a serial RS232 controller. The Component Under Test (CUT) is the TRAPP-2 software.

The workload parameters include the type of BitTorrent, SIP, DNS, and non-BitTorrent/SIP/DNS packet, as well as a network load. The single system parameter is the hash list size. The metrics include the packet processing time and the probability of packet intercept.

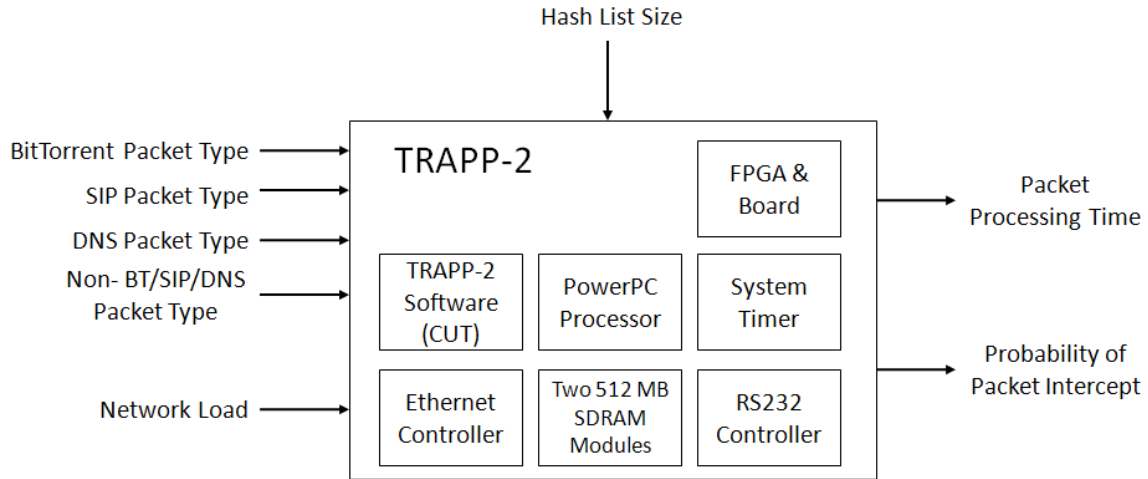


Figure 3.3: The TRacking and Analysis for Peer-to-Peer 2 System Under Test.

### 3.4 *System Services*

The TRAPP-2 system assists network administrators, law enforcement officials, and intelligence agencies in detecting and tracking traffic of interest. The system resides between a local area network and the Internet gateway and receives all the traffic flowing through the gateway. Figure 3.2 illustrates the functionality of the TRAPP-2 system.

The system is successful when the following steps are all completed:

1. A BitTorrent handshake packet, a SIP INVITE or BYE packet, or a DNS request is detected.
2. The respective file info hash, SIP URIs, or DNS domain are extracted.
3. (SIP/DNS only) The domain is sdbm hashed.
4. The hash is compared against separate lists of interest for BitTorrent, SIP, and DNS packets.
5. If a match is found for the BitTorrent and SIP packets, the packet contents are written to a Wireshark-compatible log file. For DNS, if the DNS hash is *not* found on the DNS hash whitelist, the packet is written to a Wireshark-compatible log file.

A system service failure occurs when:

1. The system does not detect a packet of interest when one is present.
2. A packet of interest is detected but the file info hash, SIP URIs, or DNS domain are not extracted.
3. The packet information is not written to the log file.

As with the TRAPP-1 system, false positives are not considered. The assumption is that an administrator will review the contents of the log file to validate the packets of interest [Sch09]. For BitTorrent, SIP, and DNS, a 4-byte hash is compared against the hash list resulting in a probability of collision of 1 in 4,294,967,296 ( $2^{32}$ ).

The TRAPP-2 system can be configured to compare larger hashes to decrease false positives and the probability of collision.

### **3.5 Workload**

The workload for the TRAPP-2 SUT consists of BitTorrent/SIP/DNS packets, a non-BitTorrent/SIP/DNS packet, and a network load. To reduce the number of packet type factors, specific packets are selected for the BitTorrent/SIP/DNS workload. Each protocol has a worst- and best-case packet for packets of interest, and only a worst-case packet for uninteresting packets. This allows a range of packet processing times to be established by using the extremes for each protocol.

Additionally, it is important to generate packets on the network that are BitTorrent, SIP, or DNS packets, but not of interest. These packets ensure that the system detects the three protocols, but the hash is not of interest. For uninteresting packets, only the worst-case scenario packet is selected.

For each of the protocols, a weighted system is used to select the worst- and best-case scenarios for each packet type. Certain characteristics of each packet determine the effectiveness of the TRAPP-2 system. Packets acquire points for having certain characteristics. For each type of protocol, the packets with the least amount of points (best-case) and most amount of points (worst-case) are used. The points are further explained under each characteristic. The characteristics used for packet selection include:

1. Hash is on/off the hash list
2. The location of the hash on the hash list
3. (SIP/DNS only) The size of the packet in bytes
4. (SIP/DNS only) The length of the domain required to sdbm hash



### 3.5.1 Packet Workload Characteristics.

*3.5.1.1 Hash Is On/Off the Hash List.* If a BitTorrent or SIP packet has a hash on the blacklist, then it must be logged, which requires additional CPU cycles. If a DNS packet does not have a hash on the whitelist, then it must be logged, which requires additional CPU cycles. For the weighted system, a packet gets 1 point for having a hash on the hash list (for BitTorrent and SIP) or 1 point for having a hash off the hash list (for DNS).

*3.5.1.2 Location of Hash on the Hash List.* The location of the hash on the hash list affects the number of CPU cycles used by the binary search algorithm. If the hash is in the middle of the list, the binary search algorithm finds it on the first try, thus requiring the fewest CPU cycles. If the hash is at the end of the list, or off the list, the algorithm requires the most comparisons, and thus more CPU cycles, to locate the hash. For the weighted system, a packet gets 1 point if its hash is at worst possible location on the hash list (for BitTorrent and SIP) or 1 point if it is off the list (for DNS).

*3.5.1.3 SIP/DNS Only: Size of the Packet.* Since the TRAPP-2 system must copy the entire packet into a software buffer, the size of the packet affects how quickly this is accomplished. Pilot tests reveal that 67-byte packets averaged 999 CPU cycles and 1500-byte packets averaged 18,112 CPU cycles. Thus, the size of the packet impacts the amount of CPU cycles required to process it. For the weighted system, a packet gets 1 point for being the largest. BitTorrent handshake packets do not exceed 122 bytes so only one packet size is used. The pilot test data for the packet size transfer times can be found in Appendix B.

*3.5.1.4 SIP/DNS Only: Length of Domain.* Since the sdbm hashing function is utilized in the TRAPP-2 system, the domain length of SIP URIs and DNS requests affect the number of CPU cycles required to generate the four-byte hash.

Refer to Section 3.2.2 for the specific numbers. For the weighted system, a SIP/DNS packet gets 1 point if it has the largest possible domain.

*3.5.2 BitTorrent Workload.* The different possible types of BitTorrent packets are illustrated in Figure 3.4. The hierarchy is read from left to right. One type of packet, for example, is a BitTorrent packet with a hash on the hash list, with a hash located in the worst hash list location. This can be abbreviated as BT-ON-WORST. Unlike SIP or DNS packets, the size of the BitTorrent handshake packet does not exceed 122 bytes, so the packet size characteristic is eliminated. See Section 2.1.1 for more details about the contents of the BitTorrent handshake packet.

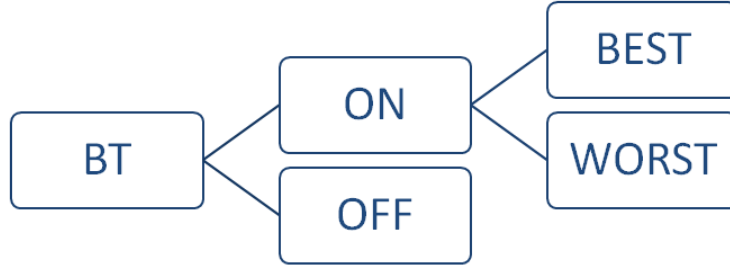


Figure 3.4: BitTorrent Packet Type Hierarchy for the TRacking and Analysis for Peer-to-Peer 2 System.

Using the weighted packet system, the scores for each type of BitTorrent packet are in Table 3.1, with the selected BitTorrent workload packets in bold. In the case of BitTorrent packets on the list, the best-case scenario packet is the BT-ON-BEST, with a score of 1. The worst-case scenario packet is the BT-ON-WORST, with a score of 2. For BitTorrent packets not on the list, the worst-case scenario packet is BT-OFF with a score of 0. So, in the case of BitTorrent packets, all three combinations of BitTorrent packets are used.

Table 3.1: BitTorrent Packet Weights for the TRacking and Analysis for Peer-to-Peer 2 System.

Protocol	On/Off Hash List		Best/Worst Hash Location		Total
<b>BT</b>	<b>ON</b>	<b>1</b>	<b>BEST</b>	<b>0</b>	<b>1</b>
	<b>ON</b>	<b>1</b>	<b>WORST</b>	<b>1</b>	<b>2</b>
	<b>OFF</b>	<b>0</b>	<b>-</b>	<b>0</b>	<b>0</b>

**BOLD** = Selected BitTorrent Workload Packet

To summarize, the three types of BitTorrent packets used are:

1. BT-ON-BEST: A 122-byte BitTorrent packet with a hash on the hash list at the best location (middle of the hash list).
2. BT-ON-WORST: A 122-byte BitTorrent packet with a hash on the hash list at the worst location (beginning of the hash list).
3. BT-OFF: A 122-byte BitTorrent packet with a hash not on the hash list.

*3.5.2.1 BitTorrent Workload Packets.* Three types of BitTorrent packets are used. The contents of BT-ON-BEST:

```

00 1c 23 18 d9 db 00 1c 23 0f 6e c9 08 00 45 00  ..#.....#.n...E.
00 6c 0c 1a 40 00 80 06 6b 1e c0 a8 01 02 c0 a8  .l...@...k.....
01 01 04 64 e8 84 65 63 a1 48 4c 7d 0b 05 50 18  ...d...ec.HL}...P.
ff ff 7e 9d 00 00 13 42 69 74 54 6f 72 72 65 6e  ..~....BitTorren
74 20 70 72 6f 74 6f 63 6f 6c 00 00 00 00 00 10  t protocol.....
00 05 68 37 67 65 b8 c6 60 98 5b df 3c 30 cd bf  ..h7ge..'.[.<0..
e5 7d fd 36 76 13 2d 55 54 32 30 30 30 2d 00 46  .}.6v.-UT2000-.F
6c e5 aa e6 bc 6a d0 02 58 95                      l....j...X.

```

The contents of BT-ON-WORST:

```

00 1c 23 18 d9 db 00 1c 23 0f 6e c9 08 00 45 00  ..#.....#.n...E.
00 6c 0c 1a 40 00 80 06 6b 1e c0 a8 01 02 c0 a8  .l...@...k.....
01 01 04 64 e8 84 65 63 a1 48 4c 7d 0b 05 50 18  ...d...ec.HL}...P.
ff ff b7 9d 00 00 13 42 69 74 54 6f 72 72 65 6e  ....BitTorren
74 20 70 72 6f 74 6f 63 6f 6c 00 00 00 00 00 10  t protocol.....
00 05 30 30 66 6c b8 c6 60 98 5b df 3c 30 cd bf  ..00fl..'.[.<0..
e5 7d fd 36 76 13 2d 55 54 32 30 30 30 2d 00 46  .}.6v.-UT2000-.F
6c e5 aa e6 bc 6a d0 02 58 95                      l....j...X.

```

The contents of BT-OFF:

```

00 1c 23 18 d9 db 00 1c 23 0f 6e c9 08 00 45 00  ..#.....#.n...E.
00 6c 0c 1a 40 00 80 06 6b 1e c0 a8 01 02 c0 a8  .l...@...k.....
01 01 04 64 e8 84 65 63 a1 48 4c 7d 0b 05 50 18  ...d...ec.HL}..P.
ff ff b4 fa 00 00 13 42 69 74 54 6f 72 72 65 6e  ....BitTorren
74 20 70 72 6f 74 6f 63 6f 6c 00 00 00 00 00 10 t protocol.....
00 05 d0 66 c8 d8 b8 c6 60 98 5b df 3c 30 cd bf  ...f....'..[.<0..
e5 7d fd 36 76 13 2d 55 54 32 30 30 30 2d 00 46  .}.6v.-UT2000-.F
6c e5 aa e6 bc 6a d0 02 58 95                    l....j..X.

```

*3.5.3 SIP Workload.* The different SIP packet type combinations are illustrated in Figure 3.5. The hierarchy is read from left to right. One type of packet, for example, is a SIP INVITE packet with a hash on the hash list, small in byte size, with a hash located in the best hash list location. This can be abbreviated as SIP-INVITE-ON-SMALL-BEST.

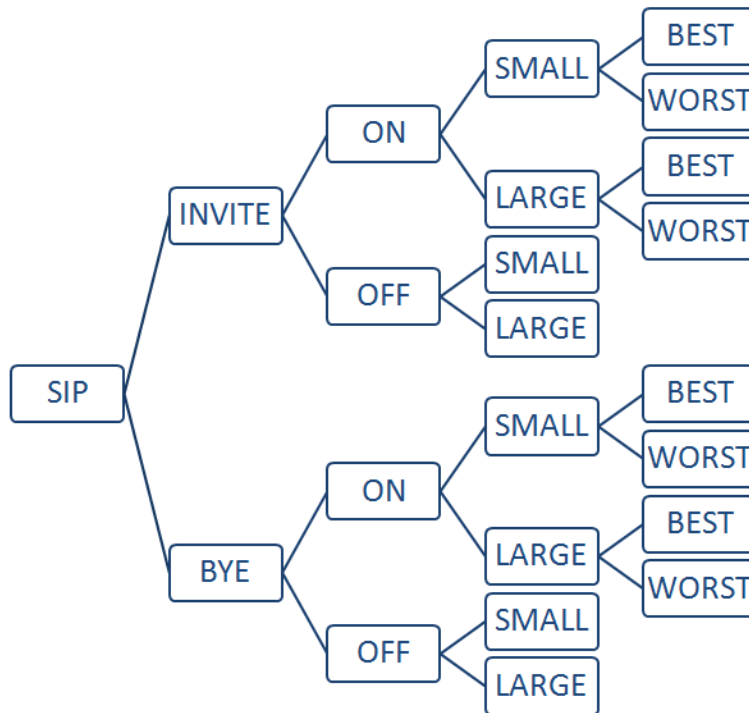


Figure 3.5: Session Initiation Protocol Packet Type Hierarchy for the TRacking and Analysis for Peer-to-Peer 2 System.

Using the weighted packet system, the scores for each type of SIP INVITE packet are in Table 3.2, with the selected SIP INVITE workload packets in bold. In the case of SIP INVITE packets on the list, the best-case scenario packet is the SIP-INVITE-ON-SMALL-BEST, with a score of 1. The worst-case scenario for a SIP INVITE packet is the SIP-INVITE-ON-LARGE-WORST, with a score of 3. For SIP INVITE packets not on the list, the worst-case scenario packet is the SIP-INVITE-OFF-LARGE with a score of 1.

Table 3.2: Session Initiation Protocol INVITE Packet Weights for the TRacking and Analysis for Peer-to-Peer 2 System.

Protocol	INV/BYE	On/Off Hash List		Packet Size		Best/Worst Hash Location		Total
<b>SIP</b>	<b>INV</b>	<b>ON</b>	<b>1</b>	<b>SMALL</b>	<b>0</b>	<b>BEST</b>	<b>0</b>	<b>1</b>
	INV	ON	1	SMALL	0	WORST	1	2
	INV	ON	1	LARGE	1	BEST	0	2
	<b>INV</b>	<b>ON</b>	<b>1</b>	<b>LARGE</b>	<b>1</b>	<b>WORST</b>	<b>1</b>	<b>3</b>
	INV	OFF	0	SMALL	0	-	-	0
	<b>INV</b>	<b>OFF</b>	<b>0</b>	<b>LARGE</b>	<b>1</b>	-	-	<b>1</b>

**BOLD** = Selected SIP INVITE Workload Packet

As a reminder, the TRAPP-2 system is looking for both SIP INVITE and SIP BYE packets. Using the weighted packet system for SIP BYE packets, the scores for each type are in Table 3.3, with the selected SIP BYE workload packets in bold. The best-case scenario for SIP BYE packets is the SIP-BYE-ON-SMALL-BEST, with a score of 1. The worst-case scenario for SIP BYE packets is the SIP-BYE-ON-LARGE-WORST, with a score of 3. The SIP BYE packet with a URI not on the list (OFF) is omitted for two reasons. The first is that from the viewpoint of the TRAPP-2 software, the packets are the same. Secondly, the SIP INVITE will take longer to process because of the larger packet size and thus represents the worst-case between the two.

Table 3.3: Session Initiation Protocol BYE Packet Weights for the TRacking and Analysis for Peer-to-Peer 2 System.

Protocol	INV/BYE	On/Off Hash List		Packet Size		Best/Worst Hash Location		Total
<b>SIP</b>	<b>BYE</b>	<b>ON</b>	<b>1</b>	<b>SMALL</b>	<b>0</b>	<b>BEST</b>	<b>0</b>	<b>1</b>
	BYE	ON	1	SMALL	0	WORST	1	2
	BYE	ON	1	LARGE	1	BEST	0	2
	<b>BYE</b>	<b>ON</b>	<b>1</b>	<b>LARGE</b>	<b>1</b>	<b>WORST</b>	<b>1</b>	<b>3</b>

**BOLD** = Selected SIP BYE Workload Packet

To summarize, the five types of SIP packets used are:

1. SIP-INVITE-ON-SMALL-BEST: A 932-byte SIP INVITE packet with a hash on the hash list at the best location.
2. SIP-INVITE-ON-LARGE-WORST: A 1500-byte SIP INVITE packet with a hash on the hash list at the worst location.
3. SIP-INVITE-OFF-LARGE: A 1500-byte SIP INVITE packet with a hash not on the hash list.
4. SIP-BYE-ON-SMALL-BEST: A 479-byte SIP BYE packet with a hash on the hash list at the best location.
5. SIP-BYE-ON-LARGE-WORST: A 1040-byte SIP BYE packet with a hash on the hash list at the worst location.

*3.5.3.1 SIP Workload Packets.* Five types of SIP packets are used.

The length of the domain affects the size of the SIP packet. For small packets, the domain 192.168.3.110 is used. For large packets, the artificially created domain below is used:

```
abcdefghijklmnopqrstuvwxyz.abcdefghijklmnopqrstuvwxyz.abcdefgh
ijklmnopqrstuvwxyz.abcdefghijklmnopqrstuvwxyz.abcdefghijklmnopqrstuvwxyz
qrstuvwxyz.abcdefghijklmnopqrstuvwxyz.abcdefghijklmnopqrstuvwxyz
yz.com.localhost
```

The contents of SIP-INVITE-ON-SMALL-BEST:

```
INVITE sip:2000@192.168.3.110 SIP/2.0
Via: SIP/2.0/UDP 192.168.1.3:39966;branch=z9hG4bK-d8754z-
be20982fbb7fb76f-1---d8754z-;rport
Max-Forwards: 70
Contact: <sip:2001@192.168.1.3:39966>
To: "2000"<sip:2000@192.168.3.110>
From: "Beta"<sip:2001@192.168.3.110>;tag=5c4c0451
Call-ID: MGIwNTdiMDI5NzU2YzhmMDEzYzMzMzU2Y2QzOWRhODQ.
CSeq: 1 INVITE
Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER, NOTIFY,
MESSAGE, SUBSCRIBE, INFO
Content-Type: application/sdp
User-Agent: X-Lite release 1104o stamp 56125
Content-Length: 360
```

The contents of SIP-INVITE-ON-LARGE-WORST:

```
INVITE sip:2001@abcdefghijklmnopqrstuvwxyz.abcdefghijklmnopqrstuvwxyz.abc
defghijklmnopqrstuvwxyz.abcdefghijklmnopqrstuvwxyz.abcdefghijklmnopqrstuvwxyz.abc
defghijklmnopqrstuvwxyz.abcdefghijklmnopqrstuvwxyz.abcdefghijklmnopqrstuvwxyz.com.local
host SIP/2.0
Via: SIP/2.0/UDP 192.168.1.2:57538;branch=z9hG4bK-d8754z-
a160be13fe074026-1---d8754z-;rport
Max-Forwards: 70
Contact: <sip:2000@192.168.1.2:57538>
To: "2001"<sip:2001@abcdefghijklmnopqrstuvwxyz.abcdefghijklmnopqrstuvwxyz.abc
defghijklmnopqrstuvwxyz.abcdefghijklmnopqrstuvwxyz.abcdefghijklmnopqrstuvwxyz.abc
defghijklmnopqrstuvwxyz.abcdefghijklmnopqrstuvwxyz.com.localhost>
From: "Alpha"<sip:2000@abcdefghijklmnopqrstuvwxyz.abcdefghijklmnopqrstuvwxyz.abc
defghijklmnopqrstuvwxyz.abcdefghijklmnopqrstuvwxyz.abcdefghijklmnopqrstuvwxyz.abc
defghijklmnopqrstuvwxyz.abcdefghijklmnopqrstuvwxyz.com.localhost>;tag=8356d139
Call-ID: M2UwNmQ3MDVlNjcOMzA5ODE4ZmFlMWU2ZmU2MzhiMWI.
CSeq: 1 INVITE
Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER, NOTIFY, MESSAGE,
SUBSCRIBE, INFO
Content-Type: application/sdp
User-Agent: X-Lite release 1104o stamp 56125
Content-Length: 360
```

The contents of SIP-INVITE-OFF-LARGE (note that 1234 replaced abcd for the first four characters of the domain):

```
INVITE sip:2001@1234efghijklmnopqrstuvwxyz.abcdefghijklmnopqrstuvwxyz.abcdefghijklmnopqrstuvwxyz.abcdefghijklmnopqrstuvwxyz.abcdefghijklmnopqrstuvwxyz.com.localhost SIP/2.0
Via: SIP/2.0/UDP 192.168.1.2:57538;branch=z9hG4bK-d8754z-a160be13fe074026-1---d8754z-;rport
Max-Forwards: 70
Contact: <sip:2000@192.168.1.2:57538>
To: "2001"<sip:2001@1234efghijklmnopqrstuvwxyz.abcdefghijklmnopqrstuvwxyz.abcdefghijklmnopqrstuvwxyz.abcdefghijklmnopqrstuvwxyz.abcdefghijklmnopqrstuvwxyz.com.localhost>
From: "Alpha"<sip:2000@1234efghijklmnopqrstuvwxyz.abcdefghijklmnopqrstuvwxyz.abcdefghijklmnopqrstuvwxyz.abcdefghijklmnopqrstuvwxyz.com.localhost>;tag=8356d139
Call-ID: M2UwNmQ3MDVlNjcOMzA5ODE4ZmFlMWU2ZmU2MzhiMWI.
CSeq: 1 INVITE
Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER, NOTIFY, MESSAGE, SUBSCRIBE, INFO
Content-Type: application/sdp
User-Agent: X-Lite release 1104o stamp 56125
Content-Length: 360
```

The contents of SIP-BYE-ON-SMALL-BEST:

```
BYE sip:2001@192.168.1.3:39966 SIP/2.0
Via: SIP/2.0/UDP 192.168.3.110:5060;branch=z9hG4bK14b4f667;rport
Max-Forwards: 70
From: "2000"<sip:2000@192.168.3.110>;tag=as758a70a9
To: "Beta"<sip:2001@192.168.3.110>;tag=5c4c0451
Call-ID: MGIwNTdiMDI5NzU2YzhmMDEzYzIxMzU2Y2QzOWRhODQ.
CSeq: 102 BYE
User-Agent: Asterisk PBX 1.6.0.10-FONCORE-r40
X-Asterisk-HangupCause: Normal Clearing
X-Asterisk-HangupCauseCode: 16
Content-Length: 0
```



The contents of SIP-BYE-ON-LARGE-WORST:

```
BYE sip:2001@192.168.1.5 SIP/2.0
Via: SIP/2.0/UDP 192.168.1.2:57538;branch=z9hG4bK-d8754z-
014fc825e962763e-1---d8754z-;rport
Max-Forwards: 70
Contact: <sip:2000@192.168.1.2:57538>
To: "2001"<sip:2001@abcdefghijklmnopqrstuvwxyz.abcdefghijklmnop
pqrstuvwxyz.abcdefghijklmnopqrstuvwxyz.abcdefghijklmnopqrstu
vwxyz.abcdefghijklmnopqrstuvwxyz.abcdefghijklmnopqrstuvwxyz.abcd
efghijklmnopqrstuvwxyz.com.localhost>;tag=as22d800eb
From: "Alpha"<sip:2000@abcdefghijklmnopqrstuvwxyz.abcdefghijklmnop
mnopqrstuvwxyz.abcdefghijklmnopqrstuvwxyz.abcdefghijklmnopqrst
uvwxyz.abcdefghijklmnopqrstuvwxyz.abcdefghijklmnopqrstuvwxyz.a
bcdefghijklmnopqrstuvwxyz.com.localhost>;tag=8356d139
Call-ID: M2UwNmQ3MDVlNjcOMzA5ODE4ZmFlMWU2ZmU2MzhiMWI.
CSeq: 3 BYE
User-Agent: X-Lite release 1104o stamp 56125
Authorization: Digest username="2000",realm="asterisk",nonce="669dc6aa",
uri="sip:2001@192.168.1.5",response="91d3092c48e35c3275c4f2f47e57336d",
algorithm=MD5
Reason: SIP;description="User Hung Up"
Content-Length: 0
```

*3.5.4 DNS Workload.* The different possible types of DNS packets are illustrated in Figure 3.6. The hierarchy is read from left to right. One type of packet, for example, is a DNS packet with a hash on the whitelist, large in byte size, with a hash located in the worst location. This can be abbreviated as DNS-ON-LARGE-WORST.

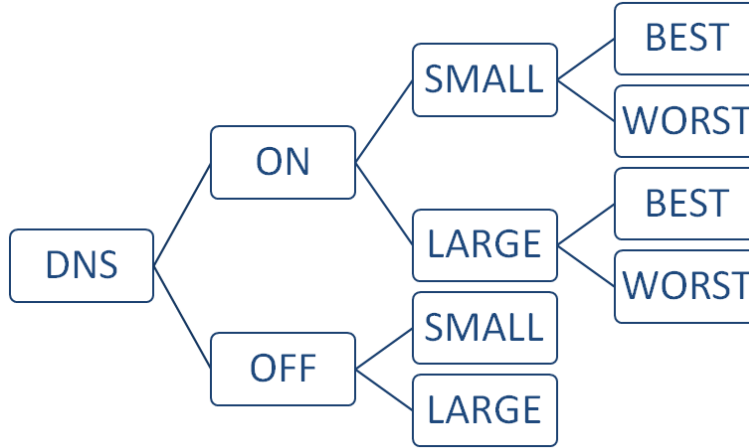


Figure 3.6: Domain Name System Packet Type Hierarchy for the TRacking and Analysis for Peer-to-Peer 2 System.

Using the weighted packet system, the scores for each type of DNS packet are in Table 3.4, with the selected DNS workload packets in bold. For DNS packets with hashes not on the whitelist (the ones of interest), the best-case scenario is the DNS-OFF-SMALL packet, with a score of 1. The worst-case scenario packet is the DNS-OFF-LARGE packet, with a score of 2. For valid DNS domains on the whitelist, the worst-case scenario is the DNS-ON-LARGE-WORST packet, with a score of 2.

Table 3.4: Domain Name System Packet Weights for the TRacking and Analysis for Peer-to-Peer 2 System.

Protocol	On/Off hash list		Packet Size		Best/Worst hash location		Total
<b>DNS</b>	<b>OFF</b>	<b>1</b>	<b>SMALL</b>	<b>0</b>	-	<b>1</b>	<b>1</b>
	<b>OFF</b>	<b>1</b>	<b>LARGE</b>	<b>1</b>	-	<b>1</b>	<b>2</b>
	ON	0	SMALL	0	BEST	0	0
	ON	0	SMALL	0	WORST	1	1
	ON	0	LARGE	1	BEST	0	1
	<b>ON</b>	<b>0</b>	<b>LARGE</b>	<b>1</b>	<b>WORST</b>	<b>1</b>	<b>2</b>

**BOLD** = Selected DNS Workload Packet

To summarize, the three types of DNS packets used are:

1. DNS-OFF-SMALL: A 67-byte DNS request packet with a hash not on the hash list
2. DNS-OFF-LARGE: A 190-byte DNS request packet not on the hash list. The packet is generated using Iodine [Kry09], a DNS exfiltration tool. The authenticity of the malicious packet is selected over a larger actual byte size, which can be achieved by increasing the domain length.
3. DNS-ON-LARGE-WORST: A 190-byte DNS request packet with a hash on the hash list in the worst location

*3.5.4.1 DNS Workload Packets.* Three types of DNS packets are used.

The contents of DNS-OFF-SMALL:

```
00 0f 1f 69 b9 87 00 1e ec f2 99 ca 08 00 45 00 ...i.....E.
00 35 39 3c 40 00 40 11 e9 18 0a 01 02 5c 0a 01 .59<@.@.....\..
02 06 c2 a3 00 35 00 21 64 cd d7 ff 01 00 00 01 .....5.!d.....
00 00 00 00 00 00 03 63 6e 6e 03 63 6f 6d 00 00 .....cnn.com..
01 00 01 ...
```

The contents of DNS-OFF-LARGE:

```
00 1f 3b 81 3f b7 00 0c 41 78 26 63 08 00 45 00 ...;.?...Ax&c..E.
00 b0 e4 8d 40 00 7d 11 8c 56 47 40 91 da d9 d9 ....@.}.VG@....
d9 64 11 5c 00 35 00 9c 4d 09 02 8b 01 00 00 01 .d.\.5..M.....
00 00 00 00 00 01 3d 30 61 64 62 45 6e 50 4a 79 .....=0adbEnPJy
67 72 47 43 67 76 47 53 68 4e 73 5a 43 64 71 57 grGCgvGShNsZCdqW
70 48 70 43 69 2d 61 61 58 6d 71 6d 32 69 57 4d pHpCi-aaXmqm2iWM
6d 41 57 6d 69 47 57 6a 63 79 4c 50 35 73 4d 50 mAWmiGWjcyLP5sMP
77 44 4b 7a 1b 4d 78 4e 34 6f 42 4c 66 58 71 77 wDKz.MxN4oBLfXqw
66 7a 45 75 4c 50 78 64 48 71 68 5a 5a 72 6c 56 fzEuLPxdHqhZZr1V
08 72 65 73 65 61 72 63 68 10 72 61 6e 64 6f 6d .research.random
68 61 63 6b 65 72 73 69 74 65 03 63 6f 6d 00 00 hackersite.com..
0a 00 01 00 00 29 10 00 00 00 80 00 00 00 .....).....
```

The contents of DNS-ON-LARGE-WORST:

```

00 1f 3b 81 3f b7 00 0c 41 78 26 63 08 00 45 00 ...;.?...Ax&c...E.
00 b0 e4 8d 40 00 7d 11 8c 56 47 40 91 da d9 d9 ....@.}.VG@....
d9 64 11 5c 00 35 00 9c d0 f5 02 8b 01 00 00 01 .d.\.5.....
00 00 00 00 00 01 03 70 69 63 74 75 72 65 73 2e .....pictures.
6d 61 69 6c 62 6f 78 2e 66 75 74 75 72 65 74 65 mailbox.futurete
63 68 6e 6f 6c 6f 67 79 64 65 73 69 67 6e 2e 74 chnologydesign.t
65 63 68 6e 69 63 61 6c 64 65 74 61 69 6c 73 67 echnicaldetailsg
72 6f 75 70 2e 73 75 70 70 6f 72 74 62 72 61 6e roup.supportbran
63 68 2e 65 6e 67 69 6e 65 65 72 69 6e 67 64 69 ch.engineeringdi
76 69 73 69 6f 6e 2e 73 75 70 65 72 6c 6f 6e 67 vision.superlong
63 6f 6d 70 61 6e 79 6e 61 6d 65 2e 63 6f 6d 00 companyname.com.
00 0a 00 01 00 00 29 10 00 00 00 80 00 00 .....).....

```

*3.5.5 Non-BitTorrent/SIP/DNS Workload.* The non-BitTorrent/SIP/DNS packet used is an HTTP packet. Its signature does not match that of a BitTorrent, SIP, or DNS packet. The HTTP packet is 389 bytes in size. The average size packet of an hour-long Wireshark network capture from a lab network with multiple computers accessing the Internet is 389 bytes. The contents of the HTTP packet:

```

00 1e 4f f2 7f 8d 00 0b fd 0d 26 a1 08 00 45 00 ..0.....&...E.
01 77 cf 35 00 00 30 06 34 1e 4a 7d 2f 65 0a 01 .w.5..0.4.J}/e..
02 4b 00 50 09 6c 3c 00 d4 25 f2 b4 eb 98 50 18 .K.P.l<..%....P.
e6 a0 e9 9f 00 00 48 54 54 50 2f 31 2e 31 20 32 .....HTTP/1.1 2
30 30 20 4f 4b 0d 0a 43 6f 6e 74 65 6e 74 2d 54 00 OK..Content-T
79 70 65 3a 20 74 65 78 74 2f 6a 61 76 61 73 63 ype: text/javasc
72 69 70 74 3b 20 63 68 61 72 73 65 74 3d 75 74 ript; charset=ut
66 2d 38 0d 0a 44 61 74 65 3a 20 54 68 75 2c 20 f-8..Date: Thu,
32 30 20 41 75 67 20 32 30 30 39 20 31 33 3a 31 20 Aug 2009 13:1
30 3a 31 34 20 47 4d 54 0d 0a 45 78 70 69 72 65 0:14 GMT..Expire
73 3a 20 54 68 75 2c 20 32 30 20 41 75 67 20 32 s: Thu, 20 Aug 2
30 30 39 20 31 34 3a 31 30 3a 31 34 20 47 4d 54 009 14:10:14 GMT
0d 0a 43 61 63 68 65 2d 43 6f 6e 74 72 6f 6c 3a ..Cache-Control:
20 70 75 62 6c 69 63 2c 20 6d 61 78 2d 61 67 65 public, max-age
3d 33 36 30 30 0d 0a 43 6f 6e 74 65 6e 74 2d 45 =3600..Content-E
6e 63 6f 64 69 6e 67 3a 20 67 7a 69 70 0d 0a 53 ncoding: gzip..S
65 72 76 65 72 3a 20 41 75 74 6f 2d 43 6f 6d 70 erver: Auto-Comp
6c 65 74 69 6f 6e 20 53 65 72 76 65 72 0d 0a 43 letion Server..C
6f 6e 74 65 6e 74 2d 4c 65 6e 67 74 68 3a 20 38 ontent-Length: 8

```

```

30 0d 0a 0d 0a 1f 8b 08 00 00 00 00 00 02 ff 2b 0.....+
cf cc 4b c9 2f d7 4b cf cf 4f cf 49 d5 4b 4c d6 ..K./..K..O.I.KL.
cb d0 88 56 ca c9 2c 4b 2d cb 4c 2d 57 28 4a 4d ...V...,K-.L-W(JM
4c d1 cd cf cb a9 54 28 2b 86 70 ca 8b 32 4b 52 L.....T(+.p..2KR
15 ca 72 cb 13 8b 94 74 a2 63 63 35 01 a4 14 cf ..r.....t.cc5....
db 41 00 00 00 .A...

```

*3.5.6 Network Load.* For Experiment 3, a network load consisting of non-BitTorrent/SIP/DNS traffic is added to the system using the Linux pktgen utility [Fou10]. By adding the load, the resulting minimum network utilization is approximately 20% and is increased at 10% intervals up to the maximum achievable rate of 93.7% (equivalent to 937 Mbps). Table 3.5 summarizes the different network utilizations achieved as a result of adding the Linux pktgen utility load. In Experiment 3, the network utilization is measured using variables within the Linux pktgen utility, described in Section 3.6.3.2.

Table 3.5: Network Utilizations Due to the Linux pktgen Utility Load in the TRacking and Analysis for Peer-to-Peer 2 System.

Mbps	Network Utilization % (Mbps/1000)
204	20.4%
301	30.1%
408	40.8%
498	49.8%
602	60.2%
714	71.4%
818	81.8%
937	93.7%

### 3.6 Performance Metrics

The two performance metrics used to evaluate the effectiveness of the TRAPP-2 system are packet processing time and the probability of packet intercept. This section also describes how the Network Utilization is measured for the experiments.

*3.6.1 Packet Processing Time.* The first metric is the packet processing time, which measures the CPU cycles used to process packets. The PowerPC's System Timer timestamp function is used for this. The packet processing time begins when a packet arrives in the Ethernet controller. Packet processing time ends immediately after processing of the packet has completed. This metric is important because packet processing time must be minimized to check every packet traversing the network. Measuring the packet processing time also gives insight into how the TRAPP-2 system responds to different packet type characteristics (Experiment 1) and hash list sizes (Experiment 4).

*3.6.2 Probability of Packet Intercept.* The second metric is the probability of packet intercept. This is calculated by determining if a packet of interest is captured and successfully recorded to the log file. When measuring the probability of packet intercept, the network utilization of the system is also measured. Experiment 2 measures the probability of packet intercept while flooding the TRAPP-2 system with protocol-under-test (BitTorrent, SIP, or DNS) traffic. The Linux utility, `tcpreplay`, is used to send a previously captured `.pcap` file containing 400 packets of interest as quickly as possible [Tcp10]. Experiment 3 measures the probability of packet intercept of packets of interest while adding a non-BitTorrent/SIP/DNS traffic load to the TRAPP-2 system. The load is generated using the Linux `pktgen` utility.

*3.6.3 Network Utilization.* The network utilization is the total amount of traffic entering the TRAPP-2 system. For Experiment 1 and Experiment 4, the network utilization is limited to single packets injected into the system, and is thus virtually zero. For Experiment 2, the network utilization varies with the type of protocol-under-test packet being flooded into the system. In Experiment 2, the network utilization is measured using Wireshark. For Experiment 3, the network utilization varies with the type of load generated by the Linux `pktgen` utility. In Experiment 3, the network utilization is measured using variables within the Linux `pktgen` utility.

*3.6.3.1 Measuring Network Utilization Using Wireshark.* For Experiment 2, the load is measured using the Wireshark laptop connected to the gigabit switch's other Switched Port Analyzer (SPAN) port. When each test has concluded, the network utilization is measured by selecting the **Statistics -> Summary** menu option and recording the **Avg. MBit/sec** from the **Displayed** column. This is assumed to be the minimum network utilization.

*3.6.3.2 Measuring the Network Utilization Using the Linux pktgen Utility Variables.* For Experiment 3, the Linux pktgen utility generates packets faster than Wireshark can process. In response to Wireshark's shortfall, an alternative method to measure the network utilization is necessary. To accomplish this, two variables must be determined to calculate the network utilization rate of megabits per second. The first is the number of bits generated on the network and the second is the amount of time elapsed to send those bits.

The Linux pktgen utility is a Bourne Again SHell (BASH) script that runs in a terminal. The Linux pktgen utility allows configuration of the packet size, number of packets, and delay. The number of packets and packet size remain static, at 6,000,000 packets and 1,500 bytes, respectively. The delay variable is modified to achieve the different network utilization percentages.

A timestamp function within the BASH scripting language is used to record the number of nanoseconds since January 1, 1970. This timestamp function is taken right before the Linux pktgen utility begins and immediately after completion. As a result, the total amount of time required to send the 6,000,000 packets is known. Since both variables are known, the megabits per second network utilization can be calculated using the formula:

$$Network\ Utilization = \frac{Packets \times Packet\ Size\ (Bytes) \times (\frac{1MB}{2^{20}B}) \times 8 \frac{Bits}{Byte}}{Elapsed\ Time}$$

Where *Packets* is 6,000,000, *Packet Size* is 1,500 and *Elapsed Time* is measured using the BASH timestamp function. A delay variable within the Linux pktgen utility BASH script is modified to throttle this rate to achieve the decreased network utilizations.

### **3.7 System Parameters**

The single TRAPP-2 system parameter is the hash list size. For Experiments 1, 2, and 3, a hash list size of 1000 is used. For Experiment 4, the hash list size is doubled from 2,000 up to 131,072,000 unique hash items. This results in 17 different hash list sizes. The hash list with 131,072,000 items is 500 MB in size, which is 97.65% of the available SDRAM memory. 100% of the memory is not used because the same memory space is used to store the log file.

The TRAPP-2 workload parameters include:

1. BitTorrent Packet Types: There are three types of BitTorrent packet types. They include BT-ON-BEST, BT-ON-WORST, and BT-OFF.
2. SIP Packet Types: There are five types of SIP packet types. They include SIP-INVITE-ON-SMALL-BEST, SIP-INVITE-ON-LARGE-WORST, SIP-INVITE-OFF-LARGE, SIP-BYE-ON-SMALL-BEST, and SIP-BYE-ON-LARGE-WORST.
3. DNS Packet Types: There are three types of DNS packet types. They include DNS-OFF-SMALL, DNS-OFF-LARGE, and DNS-ON-LARGE-WORST.
4. Non-BitTorrent/SIP/DNS Packet: This is the 389-byte HTTP packet.
5. Network Load: The additional network traffic added to the system using the Linux pktgen utility. An additional traffic load is only added in Experiment 3.

### **3.8 Factors**

For Experiments 1, 2, and 3, the TRAPP-2 software is loaded onto the FPGA board. The FPGA's Ethernet controller is configured to run at 1000 Mbps, and the



list of interest size is 1000 entries. Experiment 1 calculates the packet processing time of BitTorrent, SIP, DNS, and non-BitTorrent/SIP/DNS (the HTTP packet) packets. Experiment 2 determines the probability of packet intercept for a flood (400 packets) of protocol-under-test (BitTorrent, SIP, or DNS) traffic. The four packets selected are the worst-case for BitTorrent, SIP INVITE, SIP BYE, and DNS. Realistically, the TRAPP-2 system will not see the same packet sent as quickly as possible, but the purpose of the experiment is to stress the system. Experiment 3 determines the probability of packet intercept under the addition of eight different network loads. Table 3.6 summarizes the factor levels for Experiments 1, 2, and 3.

Table 3.6: Factor Levels for Experiments 1, 2, and 3, for the TRacking and Analysis for Peer-to-Peer 2 System.

Factor	Level 1	Level 2	Level 3	Level 4	Level 5	Level 6
Packet Type	Non BT/SIP/DNS	BT ON WORST	BT ON BEST	BT OFF	SIP INV ON SMALL BEST	SIP INV ON LARGE WORST
Approximate Network Load	None	20%	30%	40%	50%	60%

Level 7	Level 8	Level 9	Level 10	Level 11	Level 12
SIP INV OFF LARGE	SIP BYE ON SMALL BEST	SIP BYE ON LARGE WORST	DNS OFF SMALL	DNS OFF LARGE	DNS ON LARGE WORST
70%	80%	93%			

Experiment 4 tests how increasing the hash list size affects the packet processing time. The line speed of 1000 Mbps remains the same and the TRAPP-2 software is unchanged. Only large DNS packets *not* on the whitelist are used (DNS-OFF-LARGE). This ensures that the binary search algorithm is exhausted and that the packet is logged as suspicious. Table 3.7 summarizes the factor levels for Experiment 4.

Table 3.7: Factor Levels for Experiment 4 for the TRacking and Analysis for Peer-to-Peer 2 System.

Factor	Level 1	Level 2	Level 3	Level 4	Level 5	Level 6
DNS Hash List Size	2,000	4,000	8,000	16,000	32,000	64,000

Level 7	Level 8	Level 9	Level 10	Level 11	Level 12
128,000	256,000	512,000	1,024,000	2,048,000	4,096,000

Level 13	Level 14	Level 15	Level 16	Level 17
8,192,000	16,384,000	32,768,000	65,536,000	131,072,000

### 3.9 Evaluation Technique

Direct measurement is selected as the evaluation technique for the experiments because the TRAPP-2 system is a real and physical system. The experimental hardware configuration setup is on the right side of Figure 3.7. The same network is used to create the packets of interest. The packet creation configuration is on the left side of Figure 3.7. The packet creation laptops are disconnected from the network prior to conducting the experiments.

The experimental configuration consists of the following hardware:

- 1 Cisco gigabit 24-port switch (model WS-C3560G-24PS-S). The switch is configured with 22 standard ports and 2 SPAN ports.
- 1 Xilinx Virtex-5 FPGA (model FXT ML510), the SUT, connected to one of the switch's SPAN ports.
- 1 Dell Latitude D630 laptop loaded with the Windows's XP Service Pack 3 Operating System. It contains Wireshark 1.0.5 [Wir09], connected to the other switch's SPAN port, acting as the control packet sniffer. This laptop is also used to program the FPGA via Universal Serial Bus and provide Standard Input/Output for the FPGA through a RS232 interface.

- 1 Dell Latitude D630 laptop loaded with Backtrack 4 [RE10] and the tcpreplay utility, version 3.4.3 [Tcp10], to inject packets into the network.
- 1 Dell Latitude D630 loaded with the Ubuntu Desktop 9.10 Operating System. This laptop contains the Linux pktgen utility to create different network utilizations on the network.

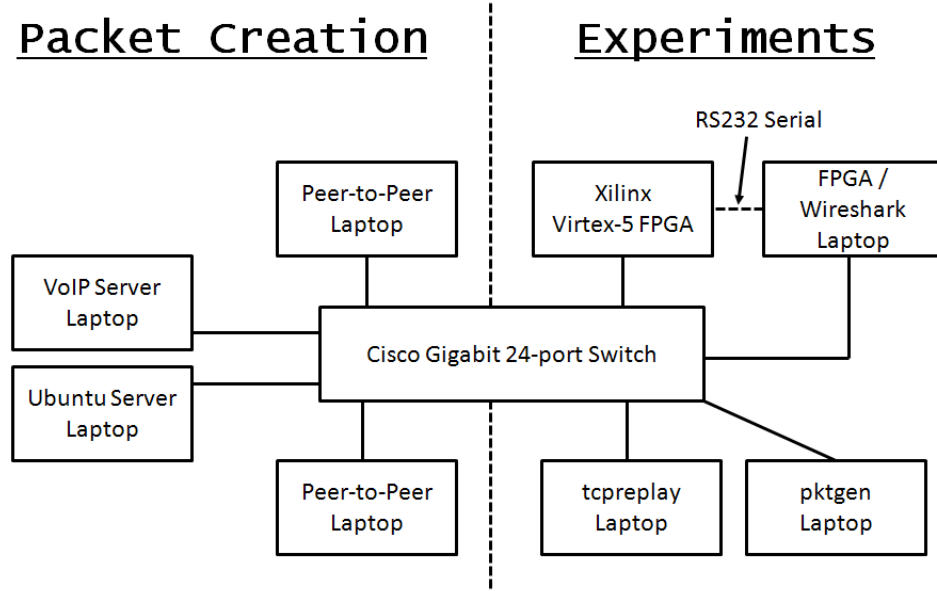


Figure 3.7: Packet Creation and Experimental Hardware Configuration Setup for the TRacking and Analysis for Peer-to-Peer 2 System.

The packet creation configuration consists of the following hardware:

- 2 Dell Latitude D630 laptops loaded with the Window's XP Service Pack 3 Operating System. They both contain uTorrent 2.0 [uTo10] and X-Lite 3.0 [Cou10], BitTorrent and VoIP clients, respectively.
- 1 Dell Latitude D630 laptop loaded with trixbox 2.8.0.3 [Tri10], based on CentOS release 5.4, acting as the SIP proxy and registrar server for the X-Lite VoIP clients.
- 1 Dell Inspiron 640m laptop loaded with Ubuntu Server 9.10 used as a DNS server. The DNS server is required to create SIP packets that contain large

domain names. This server is not used for any of the malicious DNS packet creation. The malicious DNS packets are created using Iodine [Kry09].

The actual experimental setup is shown in Figure 3.8.

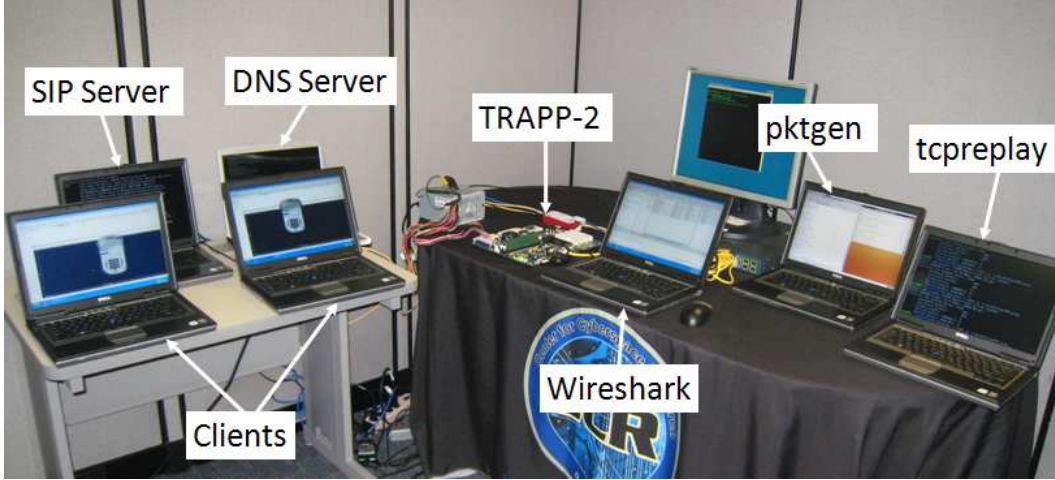


Figure 3.8: Experimental Setup for the TRacking and Analysis for Peer-to-Peer 2 System.

*3.9.1 Calculating Packet Processing Time.* For Experiment 1 and Experiment 4, a series of 50 packets is sent from the Backtrack laptop using the tcpreplay utility. Using 50 packets allows for sufficiently small confidence intervals to compare the results. For each of the three replications, a series of 50 packets is sent and the number of CPU cycles required to process the packet is recorded. Prior to sending the 50 packets, five packets are sent to the system to “warm up” the board by caching the data and instructions used by the processor. No additional network utilization is injected into the system.

*3.9.2 Calculating Probability of Packet Intercept.* For Experiment 2, a series of 400 packets is sent as fast as possible from the Backtrack laptop using the tcpreplay utility. For each of the three replications, a series of 400 protocol-under-test packets is flooded into the TRAPP-2 system and the number of packets intercepted is recorded. To stress the system and provide a sufficiently small confidence interval, 400 packets is selected. Prior to sending the 400 packets, five packets are sent to the system to

“warm up” the board by caching the data and instructions used by the processor. No additional network load is injected into the system. The network utilization is measured using Wireshark.

For Experiment 3, a series of 300 packets, sent at 200 ms intervals, is sent from the Backtrack laptop using the `tcpreplay` utility. Injecting the packets at 200 ms intervals allows for the result of each trial (captured or not captured) to be independent. The sample size of 300 packets produces a good binomial distribution with small confidence intervals. For each of the three replications, a series of 300 packets is sent into the TRAPP-2 system and the number of packets captured is recorded. Prior to sending the 300 packets, five packets are sent to the system to “warm up” the board by caching the data and instructions used by the processor. Additionally, prior to injecting the 300 packets, the Linux `pktgen` utility is activated to add the various network loads to the system. The network utilization is measured using variables within the Linux `pktgen` utility. For Experiment 2 and Experiment 3, Wireshark is used as the probability of packet intercept control.

### ***3.10 Experimental Design***

*3.10.1 Experiment 1.* Experiment 1 is a partial factorial design and calculates the packet processing time for 12 packet types and consists of 1800 trials (12 packet types x 50 packets x 3 replications). For packet processing time, a one-variable t-test is used to determine the mean packet processing time in CPU cycles, the standard deviation, the standard error of the mean, and a 95% confidence interval for the mean.

*3.10.2 Experiment 2.* Experiment 2 is a partial factorial design and calculates the probability of packet intercept for the four worst-case scenario packets: BT-ON-WORST, SIP-INVITE-ON-LARGE-WORST, SIP-BYE-ON-LARGE-WORST, and DNS-OFF-LARGE. Depending on the type of packet being investigated, the network utilization consists of either all-BitTorrent, all-SIP, or all-DNS packets that are on the

list. The test consists of 4800 trials (4 packet types x 400 packets x 3 replications). For the probability of packet intercept, a one-proportion confidence interval analysis is performed on the binomial variable to determine the probability of packet intercept and a 95% confidence interval for the proportion.

*3.10.3 Experiment 3.* Experiment 3 is a partial factorial design and calculates the probability of packet intercept for the same four worst-case scenario packet types: BT-ON-WORST, SIP-INVITE-ON-LARGE-WORST, SIP-BYE-ON-LARGE-WORST, and DNS-OFF-LARGE. However, the packets are injected into the system at 200 ms intervals. Experiment 3 is performed under eight different non-BitTorrent/SIP/DNS network utilizations, generated using the Linux pktgen utility, and consists of 28,800 trials (4 packet types x 300 packets x 8 utilizations x 3 replications). For the probability of packet intercept, a one-proportion confidence interval analysis is performed on the binomial variable to determine the probability of packet intercept and a 95% confidence interval for the proportion.

*3.10.4 Experiment 4.* Experiment 4 is a partial factorial design and calculates the packet processing time for the DNS-OFF-LARGE packet. This packet is used because it is the worst-case DNS packet. With the worst-case DNS packet, the entire hash list must be searched. Experiment 4 consists of 2,550 trials (17 list sizes x 1 packet type x 50 packets x 3 replications). For packet processing time, a one-variable t-test is used to determine the mean packet processing time in CPU cycles, the standard deviation, the standard error of the mean, and a 95% confidence interval for the mean.

### **3.11 Methodology Summary**

This section explains the experimental methods used to evaluate the performance of the TRAPP-2 system under different workloads and network utilizations. The performance is measured by calculating the packet processing time and the probability of packet intercept. Four partial factorial experiments are conducted with the

TRAPP-2 system. Experiment 1 determines the packet processing times for packets of interest. Experiment 2 determines the probability of packet intercept under a flood of 400 packets of interest. Experiment 3 determines the probability of packet intercept under various network utilizations generated using the Linux pktgen utility. Lastly, Experiment 4 determines how increasing the hash list size affects the packet processing time.

## IV. Results and Analysis

This chapter presents the results and analysis of the four experiments. Section 4.1 details the results and analysis from Experiment 1. Section 4.2 details the results and analysis from Experiment 2. Section 4.3 details the results and analysis from Experiment 3. Section 4.4 presents the results and analysis from Experiment 4. An overall analysis is provided in Section 4.5, and the chapter is summarized in Section 4.6.

### 4.1 Results and Analysis of Experiment 1

Table 4.1 summarizes the results of a one-variable t-test using 12 different packet types. The table contains the number of packets sent, mean number of CPU cycles required to process the packet, the standard deviation, the standard error of the mean, and the 95% confidence interval for the mean. The data is sorted based on the mean CPU cycles (packet processing time).

Table 4.1: Sorted Mean Packet Processing Times for Experiment 1.

Packet Type	Packets Sent	Mean CPU Cycles	Stand. Dev.	Standard Error of the Mean	Confidence Interval (95%)
DNS OFF SMALL	150	1671.83	53.33	4.35	(1663.22, 1680.43)
BT OFF	150	1973.00	0.00	0.00	(1973.00, 1973.00)
BT ON BEST	150	2085.72	52.03	4.25	(2077.33, 2094.11)
BT ON WORST	150	2217.39	50.83	4.15	(2209.19, 2225.59)
Non BT/SIP/DNS	150	4985.00	0.00	0.00	(4985.00, 4985.00)
DNS ON LARGE WORST	150	5172.00	0.00	0.00	(5172.00, 5172.00)
DNS OFF LARGE	150	5539.47	57.02	4.66	(5530.27, 5548.67)
SIP BYE ON SMALL BEST	150	8580.32	55.68	4.55	(8571.34, 8589.30)
SIP INV ON SMALL BEST	150	15283.0	545.8	44.6	(15195.0, 15371.1)
SIP BYE ON LARGE WORST	150	26071.6	655.6	53.5	(25965.9, 26177.4)
SIP INV OFF LARGE	150	31092.0	0.0	0.0	(31092.0, 31092.0)
SIP INV ON LARGE WORST	150	34226.6	879.8	71.8	(34084.6, 34368.5)

Figure 4.1 plots the sorted mean packet processing times for each of the 12 different packet types from Table 4.1. The range of mean packet processing time is



1,671.83 to 34,226.6 CPU cycles. DNS and BitTorrent packets tend to be on the lower end, while all SIP packets are on the higher end of the range.

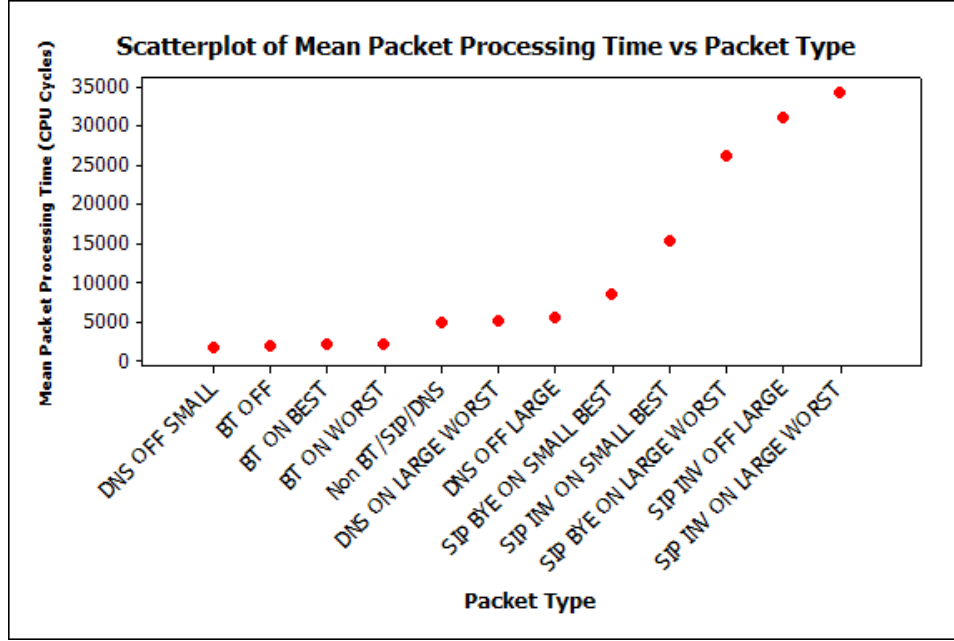


Figure 4.1: Mean Packet Processing Times for the 12 Different Packet Types for Experiment 1.

*4.1.1 BitTorrent Packet Processing Time.* Table 4.2 highlights the BitTorrent packet processing time values from Experiment 1. Figure 4.2 plots the sorted mean CPU cycles (packet processing time) and the 95% confidence intervals for the BitTorrent packets. By using the worst- and best-case scenario BitTorrent packets, a range of 2085.72 - 2,217.39 is established for all BitTorrent packets of interest.

Table 4.2: Sorted BitTorrent Mean Packet Processing Times for Experiment 1.

Packet Type	Packets Sent	Mean CPU Cycles	Stand. Dev.	Standard Error of the Mean	Confidence Interval (95%)
BT OFF	150	1973.00	0.00	0.00	(1973.00, 1973.00)
BT ON BEST	150	2085.72	52.03	4.25	(2077.33, 2094.11)
BT ON WORST	150	2217.39	50.83	4.15	(2209.19, 2225.59)

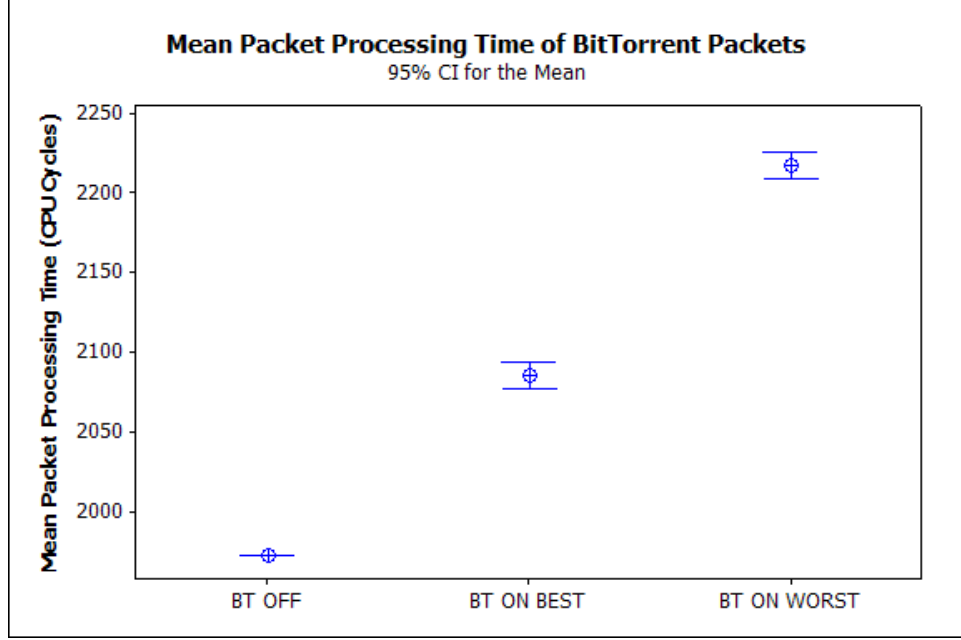


Figure 4.2: Mean Packet Processing Times for BitTorrent Packet Types for Experiment 1.

*4.1.2 SIP Packet Processing Time.* Table 4.3 highlights the SIP packet processing time values from Experiment 1. Figure 4.3 plots the sorted mean CPU cycles (packet processing time) and 95% confidence intervals for the SIP packets. By using the worst- and best-case scenario for SIP INVITE packets, a range of 15,283.0 - 34,226.6 is established for all SIP INVITE packets of interest. By using the worst- and best-case scenario for SIP BYE packets, a range of 8,580.32 - 26,071.6 is established for all SIP BYE packets of interest.

Table 4.3: Sorted Session Initiation Protocol Mean Packet Processing Times for Experiment 1.

Packet Type	Packets Sent	Mean CPU Cycles	Stand. Dev.	Standard Error of the Mean	Confidence Interval (95%)
SIP BYE ON SMALL BEST	150	8580.32	55.68	4.55	(8571.34, 8589.30)
SIP INV ON SMALL BEST	150	15283.0	545.8	44.6	(15195.0, 15371.1)
SIP BYE ON LARGE WORST	150	26071.6	655.6	53.5	(25965.9, 26177.4)
SIP INV OFF LARGE	150	31092.0	0.0	0.0	(31092.0, 31092.0)
SIP INV ON LARGE WORST	150	34226.6	879.8	71.8	(34084.6, 34368.5)

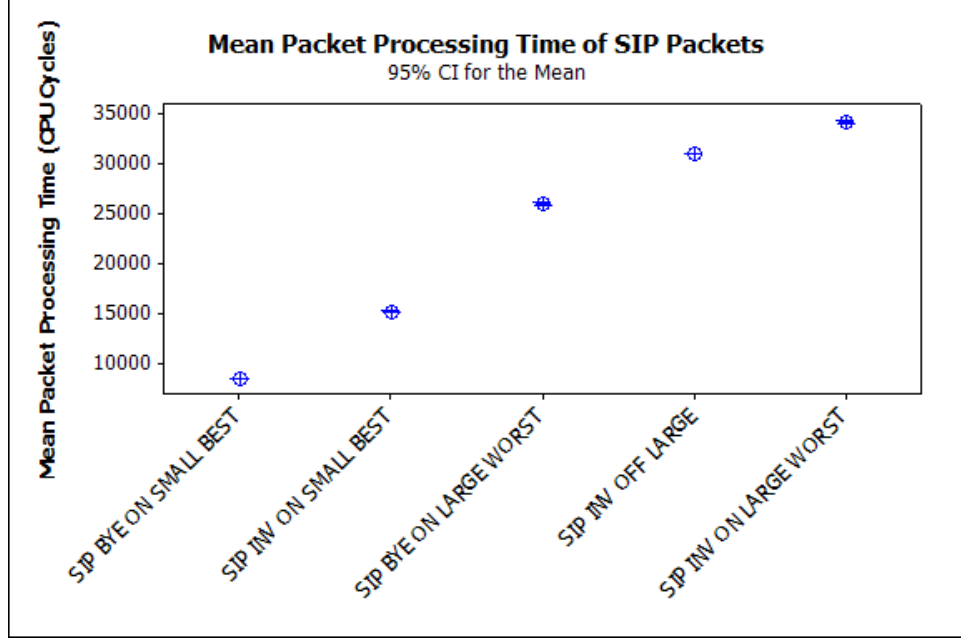


Figure 4.3: Mean Packet Processing Times for Session Initiation Protocol Packet Types for Experiment 1.

*4.1.3 DNS Packet Processing Time.* Table 4.4 highlights the DNS packet processing time values from Experiment 1. Figure 4.4 plots the sorted mean CPU cycles (packet processing time) and 95% confidence intervals for the DNS packets. By using the worst- and best-case scenario for DNS packets, a range of 1,671.83 - 5,539.47 is established for all DNS packets of interest.

Table 4.4: Sorted Domain Name System Mean Packet Processing Times for Experiment 1.

Packet Type	Packets Sent	Mean CPU Cycles	Stand. Dev.	Standard Error of the Mean	Confidence Interval (95%)
DNS OFF SMALL	150	1671.83	53.33	4.35	(1663.22, 1680.43)
DNS ON LARGE WORST	150	5172.00	0.00	0.00	(5172.00, 5172.00)
DNS OFF LARGE	150	5539.47	57.02	4.66	(5530.27, 5548.67)

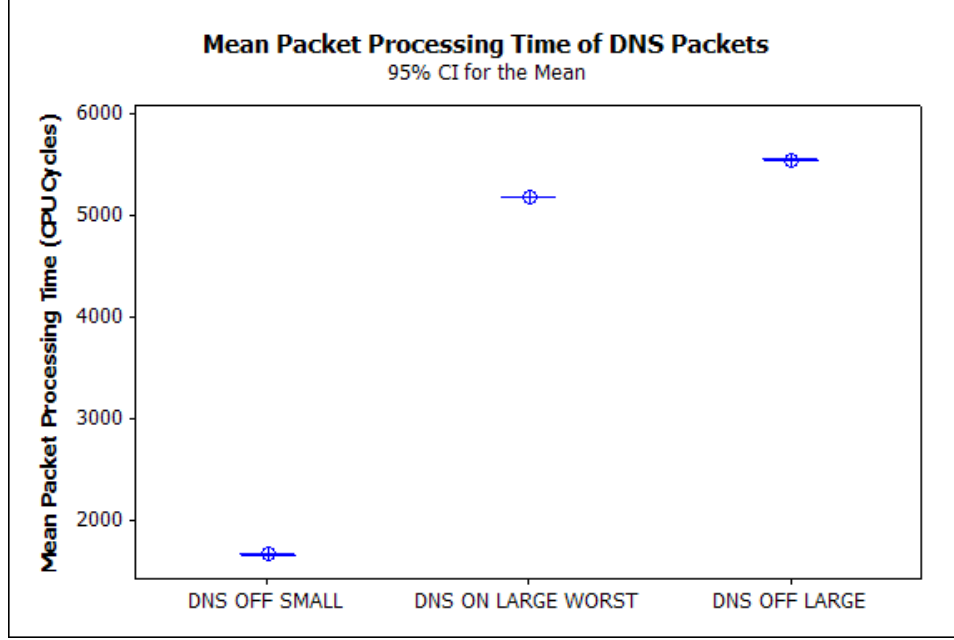


Figure 4.4: Mean Packet Processing Times for Domain Name System Packet Types for Experiment 1.

*4.1.4 Experiment 1 Analysis.* BitTorrent and DNS packets require the least amount of packet processing time. This is due to the small byte size of the packets. The packet size transfer pilot test, mentioned in Section 3.5.1.3, reveals that transferring the packet from the Ethernet buffer to a software buffer takes significantly longer for larger packets.

All five types of SIP packets require the most packet processing time. This is due to the larger byte size and processing required of SIP packets compared to BitTorrent and DNS. First, the TRAPP-2 system must copy the entire packet into a software buffer, so the larger SIP packets take longer to transfer. Secondly, the SIP packet payload must be searched for the **To:** and **From:** SIP URIs because they are not at a fixed location in the payload. Furthermore, once the **To:** and **From:** SIP URIs are extracted, they both must be separately *sdbm* hashed and searched against the hash list.

SIP BYE packets have a smaller packet processing time than SIP INVITE packets when the other packet characteristics (on/off hash list, and hash location) are equal. The disparity results because of the intrinsic larger packet size (approximately 375 bytes) of SIP INVITE packets over SIP BYE packets.

#### 4.2 Results and Analysis of Experiment 2

Table 4.5 summarizes the results of flooding the TRAPP-2 system with 400 single protocol-of-interest packets. The protocol of interest is the worst-case scenario packet for BitTorrent, SIP, and DNS. However, SIP INVITE and SIP BYE packets are tested separately even though they fall under the same protocol. As a comparison, the number of packets captured by Wireshark is also presented for each workload. The table also contains the measured network utilization, the probability of packet intercept, and the 95% confidence interval for the probability of packet intercept. The data is sorted based on the network utilization measured using Wireshark.

Table 4.5: Probability of Packet Intercept for Flood of 1200 (400 packets x 3 replications) Worst-Case Scenario Packets for Experiment 2.

Workload	Network Utilization%	Packets Captured (Events)	Packets Sent (Trials)	Prob. of Packet Intercept	Confidence Interval (95%)
BT (TRAPP-2)	15.46	1200	1200	1.0000	(0.9975, 1.0000)
BT (Wireshark)	15.46	1200	1200	1.0000	(0.9975, 1.0000)
DNS (TRAPP-2)	23.82	1039	1200	0.8658	(0.8452, 0.8846)
DNS (Wireshark)	23.82	1200	1200	1.0000	(0.9975, 1.0000)
SIP BYE (TRAPP-2)	94.75	264	1200	0.2200	(0.1969, 0.2445)
SIP BYE (Wireshark)	94.75	1200	1200	1.0000	(0.9975, 1.0000)
SIP INV (TRAPP-2)	99.48	239	1200	0.1992	(0.1769, 0.2229)
SIP INV (Wireshark)	99.48	1200	1200	1.0000	(0.9975, 1.0000)

Figure 4.5 plots the results from Table 4.5. For BitTorrent packets, both the TRAPP-2 system and Wireshark intercept 100% of the 1200 packets (400 packets x 3 replications). However, the network utilization measured during the test is only 15.46%. This is due to BitTorrent relying on TCP, which uses reliable data transfer

and exponential backoff mechanisms to throttle the throughput. For DNS packets, the TRAPP-2 system captures 86.58% of the 1200 packets, while Wireshark captures 100%. The network utilization measured during the test is 23.82%. Although DNS relies on UDP, allowing packets to be sent faster than TCP, the smaller size of the packets results in a smaller network utilization. For SIP BYE packets, the TRAPP-2 system captures 22.00% of the 1200 packets while Wireshark captures 100% of the packets. The network utilization measured during the test is 94.75%. The larger SIP BYE packet byte size and reliance on UDP results in an increased network utilization. For the final packet, SIP INVITE, the TRAPP-2 system captures 19.92% of the 1200 packets, while Wireshark again captures 100% of the packets. The 1500-byte SIP packet transferred over UDP results in the highest network utilization of 99.48%.

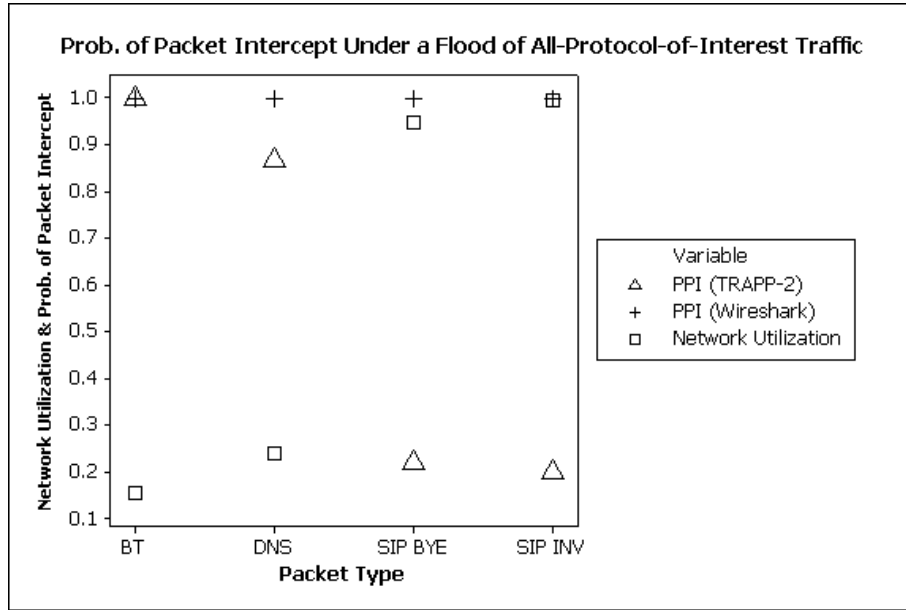


Figure 4.5: Network Utilization and Probability of Packet Intercept vs Flood of 1200 packets (400 packets x 3 replications) Worst-Case Scenario Packets for Experiment 2.

*4.2.1 Experiment 2 Analysis.* Wireshark outperforms the TRAPP-2 system in capturing a flood of 400 packets for DNS, SIP BYE, and SIP INVITE packet types. This is due to the fact that Wireshark does not perform any processing on the packets. As for the TRAPP-2 system, it must process 400 of the worst-case

scenario packets back-to-back. The results are expected since DNS, SIP BYE, and SIP INVITE packets require more CPU cycles to process than BitTorrent packets as revealed in Experiment 1. Both the TRAPP-2 system and Wireshark capture 100% of BitTorrent packets, however, the network utilization is only 15.46%. Although Wireshark outperforms the TRAPP-2 system for three of the four packets, this type of traffic is unrealistic in a real world network. For all four of these types of packets, the same packet would not typically be sent back-to-back as fast as possible.

### 4.3 Results and Analysis of Experiment 3

*4.3.1 BitTorrent Probability of Packet Intercept.* Table 4.6 presents the results for the BitTorrent packet under eight different network utilizations. As a reminder, the network utilizations are generated by adding a load using the Linux pktgen utility. The packet selected for the test is BT-ON-WORST, the worst-case scenario BitTorrent packet. As a result of selecting the BT-ON-WORST packet, the data in Table 4.6 is assumed to be the worst-case scenario for a BitTorrent packet with a hash on a hash list size of 1000. The probability of packet intercept is the percentage of 900 packets (300 packets x 3 replications) captured.

Table 4.6: Probability of Packet Intercept for BitTorrent Packets Under Various Network Utilizations for Experiment 3.

Utilization %	Prob. of Packet Intercept (TRAPP-2)	Confidence Interval (95%)	Prob. of Packet Intercept (Wireshark)	Confidence Interval (95%)
20.4%	1.0000	(0.9967, 1.0000)	0.9589	(0.9437, 0.9708)
30.1%	0.9733	(0.9605, 0.9828)	0.7167	(0.6859, 0.7459)
40.8%	0.9667	(0.9527, 0.9773)	0.4122	(0.3798, 0.4451)
49.8%	0.9711	(0.9579, 0.9810)	0.3733	(0.3416, 0.4058)
60.2%	0.9578	(0.9425, 0.9699)	0.2556	(0.2273, 0.2853)
71.4%	0.9456	(0.9286, 0.9594)	0.2356	(0.2081, 0.2646)
81.8%	0.9578	(0.9425, 0.9699)	0.2089	(0.1827, 0.2369)
93.7%	0.9556	(0.9399, 0.9680)	0.1722	(0.1481, 0.1985)

Figure 4.6 shows a plot of the probability of packet intercept for both the TRAPP-2 system and Wireshark as the network utilization is increased. The TRAPP-2 system has a higher probability of packet intercept for every network utilization level. For the 20.4% network utilization, TRAPP-2 captures 100% of BitTorrent packets and Wireshark captures 95.89%. Figure 4.6 reveals the approximate and slight linear decrease in probability of packet intercept for the TRAPP-2 system, as opposed to the exponential decrease by Wireshark, when the network utilization is increased. This is further emphasized by the fact that the TRAPP-2 system manages to capture 95.56% of BitTorrent packets at the maximum network utilization of 93.7%. In contrast, Wireshark only captures 17.22% of BitTorrent packets at the maximum network utilization.

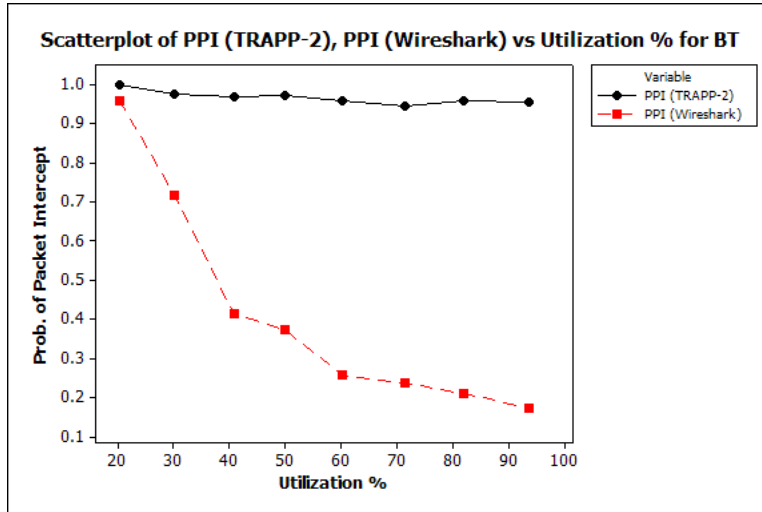


Figure 4.6: Probability of Packet Intercept for BitTorrent Packets vs Various Network Utilizations for Experiment 3.

*4.3.2 SIP INVITE Probability of Packet Intercept.* Table 4.7 presents the results for the SIP INVITE packet under eight different network utilizations. The packet selected for the test is SIP-INVITE-ON-LARGE-WORST, the worst-case scenario SIP INVITE packet. As a result of selecting the SIP-INVITE-ON-LARGE-WORST packet, the data in Table 4.7 is assumed to be the worst-case scenario for a



SIP INVITE packet with a hash on a hash list size of 1000. The probability of packet intercept is the percentage of 900 packets (300 packets x 3 replications) captured.

Table 4.7: Probability of Packet Intercept for Session Initiation Protocol INVITE Packets Under Various Network Utilizations for Experiment 3.

Utilization %	Prob. of Packet Intercept (TRAPP-2)	Confidence Interval (95%)	Prob. of Packet Intercept (Wireshark)	Confidence Interval (95%)
20.4%	1.0000	(0.9967, 1.0000)	0.9600	(0.9450, 0.9718)
30.1%	0.4256	(0.3929, 0.4586)	0.6867	(0.6552, 0.7168)
40.8%	0.3144	(0.2842, 0.3459)	0.4467	(0.4138, 0.4798)
49.8%	0.2733	(0.2444, 0.3037)	0.3478	(0.3166, 0.3799)
60.2%	0.2778	(0.2487, 0.3082)	0.2589	(0.2305, 0.2888)
71.4%	0.2522	(0.2241, 0.2819)	0.2256	(0.1986, 0.2542)
81.8%	0.2322	(0.2049, 0.2612)	0.2200	(0.1933, 0.2485)
93.7%	0.2078	(0.1817, 0.2357)	0.1633	(0.1397, 0.1891)

Figure 4.7 shows a plot of the probability of packet intercept for both the TRAPP-2 system and Wireshark as the network utilization is increased. For the 20.4% network utilization, TRAPP-2 captures 100% of SIP INVITE packets and Wireshark captures 96.00%. The TRAPP-2 system has a higher probability of packet intercept for the 20.4% network utilization, drops below Wireshark in the 30.1% - 49.8% range, then exceeds Wireshark for the remaining network utilizations. Figure 4.7 reveals the approximate exponential decrease in probability of packet intercept for both the TRAPP-2 system and Wireshark when the network utilization is increased.

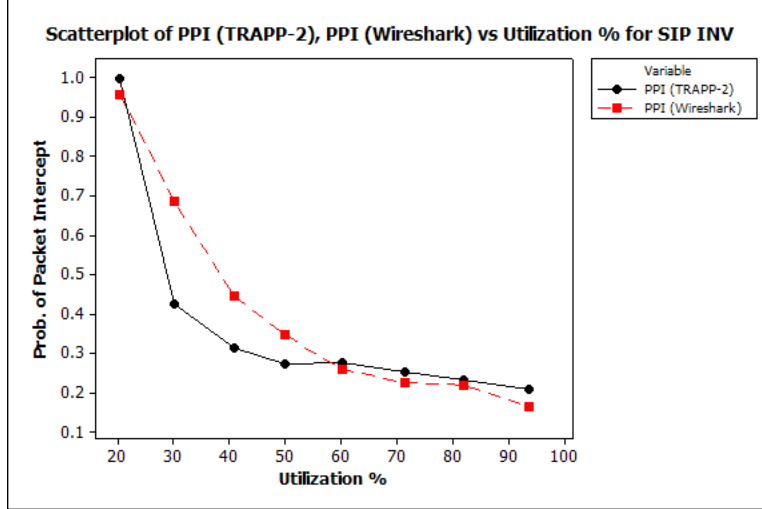


Figure 4.7: Probability of Packet Intercept for Session Initiation Protocol INVITE Packets vs Various Network Utilizations for Experiment 3.

*4.3.3 SIP BYE Probability of Packet Intercept.* Table 4.8 presents the results for the SIP BYE packet under eight different network utilizations. The packet selected for the test is SIP-BYE-ON-LARGE-WORST, the worst-case scenario SIP BYE packet. As a result of selecting the SIP-BYE-ON-LARGE-WORST packet, the data in Table 4.8 is assumed to be the worst-case scenario for a SIP BYE packet with a hash on a hash list size of 1000. The probability of packet intercept is the percentage of 900 packets (300 packets x 3 replications) captured.

Table 4.8: Probability of Packet Intercept for Session Initiation Protocol BYE Packets Under Various Network Utilizations for Experiment 3.

Utilization %	Prob. of Packet Intercept (TRAPP-2)	Confidence Interval (95%)	Prob. of Packet Intercept (Wireshark)	Confidence Interval (95%)
20.4%	1.0000	(0.9967, 1.0000)	0.9633	(0.9488, 0.9746)
30.1%	0.6344	(0.6020, 0.6659)	0.6778	(0.6461, 0.7082)
40.8%	0.5189	(0.4856, 0.5519)	0.4200	(0.3875, 0.4530)
49.8%	0.4600	(0.4270, 0.4932)	0.3078	(0.2777, 0.3390)
60.2%	0.5167	(0.4834, 0.5497)	0.3111	(0.2809, 0.3424)
71.4%	0.4711	(0.4380, 0.5043)	0.2556	(0.2273, 0.2853)
81.8%	0.4544	(0.4215, 0.4876)	0.1989	(0.1732, 0.2264)
93.7%	0.3711	(0.3394, 0.4036)	0.1589	(0.1355, 0.1844)

Figure 4.8 shows a plot of the probability of packet intercept for both the TRAPP-2 system and Wireshark as the network utilization is increased. For the 20.4% network utilization, TRAPP-2 captures 100% of SIP BYE packets and Wireshark captures 96.33%. The TRAPP-2 system has a higher probability of packet intercept for the 20.4% network utilization, drops below Wireshark at 30.1%, then exceeds Wireshark for the remaining network utilizations. Figure 4.8 reveals the approximate exponential decrease in probability of packet intercept for both the TRAPP-2 system and Wireshark when the network utilization is increased. For both the TRAPP-2 system and Wireshark, a small bump in the probability of packet intercept is visible at the 60.2% network utilization. The reason for the bump appearing in both the TRAPP-2 system and Wireshark results is unknown.

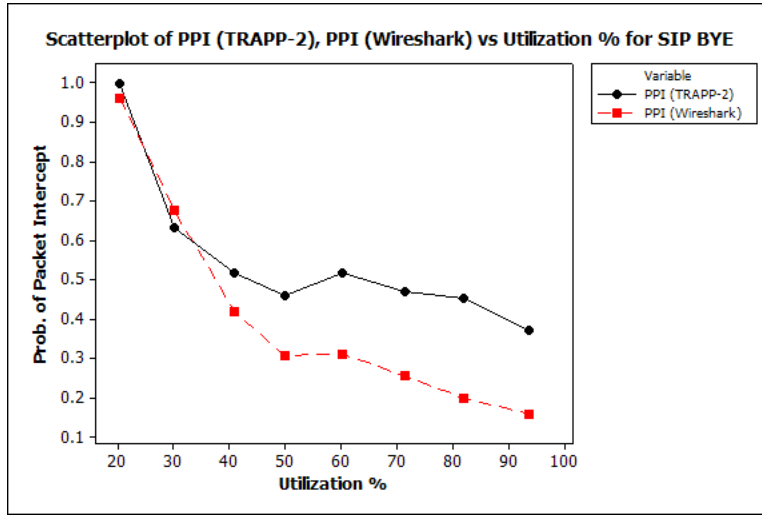


Figure 4.8: Probability of Packet Intercept for Session Initiation Protocol BYE Packets vs Various Network Utilizations for Experiment 3.

*4.3.4 DNS Probability of Packet Intercept.* Table 4.9 presents the results for the DNS packet under eight different network utilizations. The packet selected for the test is DNS-OFF-LARGE, the worst-case scenario DNS packet. As a result of selecting the DNS-OFF-LARGE packet, the data in Table 4.9 is assumed to be the worst-case scenario for a DNS packet with a hash off a hash list size of 1000.

The probability of packet intercept is the percentage of 900 packets (300 packets x 3 replications) captured.

Table 4.9: Probability of Packet Intercept for Domain Name System Packets Under Various Network Utilizations for Experiment 3.

Utilization %	Prob. of Packet Intercept (TRAPP-2)	Confidence Interval (95%)	Prob. of Packet Intercept (Wireshark)	Confidence Interval (95%)
20.4%	1.0000	(0.9967, 1.0000)	0.9567	(0.9412, 0.9690)
30.1%	0.9789	(0.9672, 0.9872)	0.6544	(0.6223, 0.6855)
40.8%	0.9644	(0.9501, 0.9755)	0.4500	(0.4171, 0.4831)
49.8%	0.9611	(0.9463, 0.9727)	0.3556	(0.3242, 0.3878)
60.2%	0.9100	(0.8893, 0.9278)	0.3144	(0.2842, 0.3459)
71.4%	0.9200	(0.9003, 0.9368)	0.2289	(0.2018, 0.2577)
81.8%	0.8911	(0.8689, 0.9107)	0.2500	(0.2220, 0.2796)
93.7%	0.9189	(0.8990, 0.9358)	0.1800	(0.1554, 0.2066)

Figure 4.9 shows a plot of the probability of packet intercept for both the TRAPP-2 system and Wireshark as the network utilization is increased. The TRAPP-2 system has a higher probability of packet intercept for every network utilization level. For the 20.4% network utilization, TRAPP-2 captures 100% of SIP BYE packets and Wireshark captures 95.67%. Figure 4.9 reveals the approximate and slight linear decrease in probability of packet intercept for the TRAPP-2 system, as opposed to the exponential decrease by Wireshark, when the network utilization is increased. This is further emphasized by the fact that the TRAPP-2 system manages to capture 91.89% of DNS packets at the maximum network utilization of 93.7%. In contrast, Wireshark only captures 18.00% of DNS packets at the maximum network utilization.

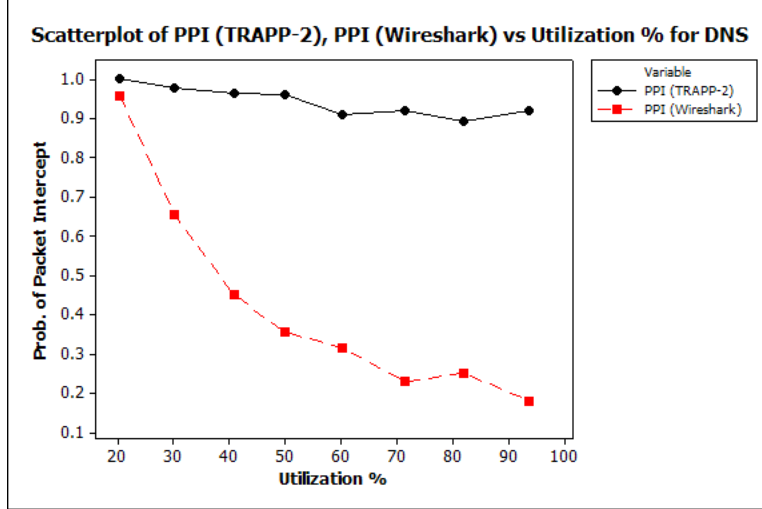


Figure 4.9: Probability of Packet Intercept for Domain Name System Packets vs Various Network Utilizations for Experiment 3.

*4.3.5 Experiment 3 Analysis.* The TRAPP-2 system significantly outperforms Wireshark in capturing BitTorrent and DNS packets. The TRAPP-2 system captures 95.56% of BitTorrent and 91.89% of DNS packets under a 93.7% network utilization. Wireshark captures 17.22% of BitTorrent and 18.00% of DNS packets under the same 93.7% network utilization. As a reminder, the packets selected for this test are the worst-case scenario packets.

The TRAPP-2 system and Wireshark return similar performances for SIP INVITE packets. However, the TRAPP-2 system captures 20.78% of packets compared to Wireshark's 16.33% under a 93.7% network utilization. The TRAPP-2 system and Wireshark return similar performances for SIP BYE packets at the 20.4% and 30.1% network utilization. After that, the TRAPP-2 system outperforms Wireshark. Under a 93.7% network utilization, the TRAPP-2 system captures 37.11% of SIP BYE packets compared to 15.89% by Wireshark. The TRAPP-2 system has a smaller probability of packet intercept for SIP INVITE and SIP BYE packets due to the larger packet size and packet processing time.

In general, the probability of packet intercept for Wireshark, regardless of packet type, follows the same exponential decrease as the network utilization is increased.

The probability of packet intercept for the TRAPP-2 system depends on the type of packet being captured. For the TRAPP-2 system, the smaller BitTorrent and DNS packets outperform the larger SIP INVITE and SIP BYE packets at higher network utilizations.

#### 4.4 Results and Analysis of Experiment 4

Table 4.10 summarizes the results of a one-variable t-test using 17 different hash list sizes. The packet selected for this test is the DNS-OFF-LARGE, the worst-case scenario DNS packet. The packet is selected because the entire hash list search is exhausted. The table contains the number of packets sent, mean number of CPU cycles required to process the DNS packet, the standard deviation, the standard error of the mean, and the 95% confidence interval for the mean.

Table 4.10: Mean Packet Processing Times for 17 Different Hash List Sizes for Experiment 4.

Hash List Items	Packets Sent	Mean CPU Cycles	Standard Deviation	Standard Error of the Mean	Confidence Interval (95%)
2,000	150	5683.87	66.14	5.40	(5673.19, 5694.54)
4,000	150	5697.71	60.08	4.91	(5688.01, 5707.40)
8,000	150	5707.06	56.62	4.62	(5697.93, 5716.19)
16,000	150	5723.72	55.12	4.50	(5714.83, 5732.61)
32,000	150	5739.69	52.16	4.26	(5731.27, 5748.10)
64,000	150	5756.57	60.68	4.95	(5746.78, 5766.36)
128,000	150	5780.12	55.10	4.50	(5771.23, 5789.01)
256,000	150	5799.23	60.81	4.97	(5789.42, 5809.04)
512,000	150	5814.21	66.36	5.42	(5803.50, 5824.91)
1,024,000	150	5830.26	77.46	6.32	(5817.76, 5842.76)
2,048,000	150	5848.29	70.11	5.72	(5836.98, 5859.61)
4,096,000	150	5867.69	73.64	6.01	(5855.81, 5879.57)
8,192,000	150	5886.57	80.20	6.55	(5873.63, 5899.51)
16,384,000	150	5901.64	81.78	6.68	(5888.45, 5914.83)
32,768,000	150	5918.90	84.62	6.91	(5905.25, 5932.55)
65,536,000	150	5931.99	52.17	4.26	(5923.58, 5940.41)
131,072,000	150	5938.81	39.85	3.25	(5932.38, 5945.24)

Figure 4.10 shows a plot of the mean packet processing time as the hash list size is increased. The smallest hash list size is 2,000 and is doubled up to 131,072,000 unique hash items on the hash list. The doubling of the hash list size results in a logarithmic plot for the mean packet processing times. Note that the difference between mean packet processing times for the hash list size of 2,000 and 131,072,000 is approximately only 255 CPU cycles.

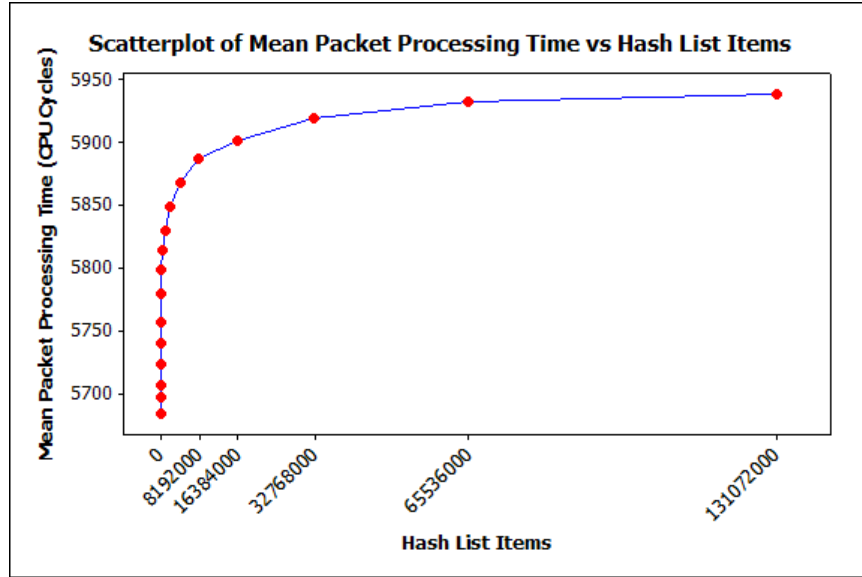


Figure 4.10: Mean Packet Processing Times vs 17 Different Hash List Sizes for Experiment 4.

To verify the logarithmic nature of the mean packet processing times as the hash list size is doubled, a separate plot is required. Figure 4.11 plots the mean packet processing times against the natural log of the hash list sizes. The linearity of the plot asserts the logarithmic nature of doubling the hash list size.

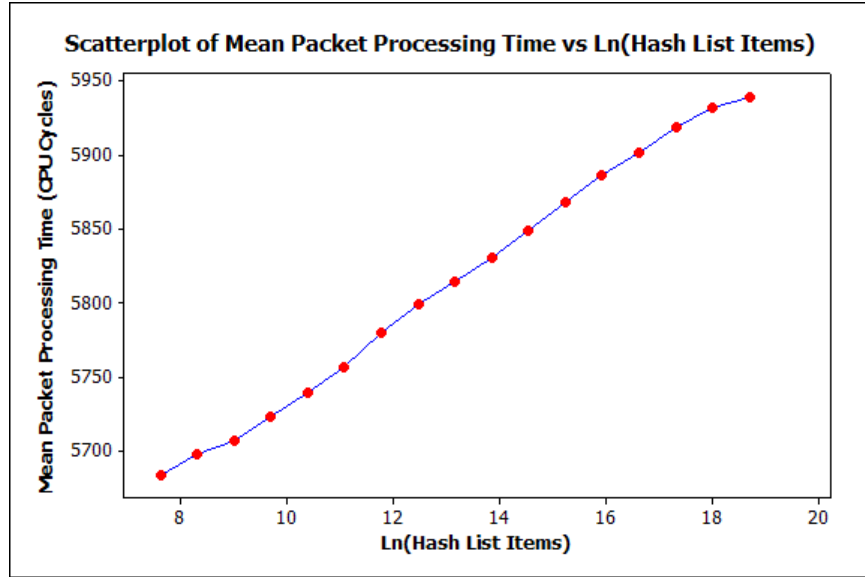


Figure 4.11: Mean Packet Processing Times vs Natural Log of 17 Different Hash List Sizes for Experiment 4.

Table 4.11 displays the mean packet processing times and the difference between them. For example, the hash list size of 4,000 takes an average of 13.84 more CPU cycles than the hash list size of 2,000. The range is from 6.22 - 23.55 CPU cycles and the overall average difference between the means is 15.93 CPU cycles. This results in the average addition of 15.93 CPU cycles to the overall packet processing time for each doubling of the hash list size.



Table 4.11: Difference Between Mean Packet Processing Times for 17 Different Hash List Sizes for Experiment 4.

Hash List Items	Mean CPU Cycles	Difference Between Means
2,000	5683.87	-
4,000	5697.71	13.84
8,000	5707.06	9.35
16,000	5723.72	16.66
32,000	5739.69	15.97
64,000	5756.57	16.88
128,000	5780.12	23.55
256,000	5799.23	19.11
512,000	5814.21	14.98
1,024,000	5830.26	16.05
2,048,000	5848.29	18.03
4,096,000	5867.69	19.40
8,192,000	5886.57	18.88
16,384,000	5901.64	15.07
32,768,000	5918.90	17.26
65,536,000	5931.99	13.09
131,072,000	5938.81	6.82
	Average Difference Between Means	15.93

*4.4.1 Experiment 4 Analysis.* Doubling the hash list size results in an average mean increase of 15.93 CPU cycles for the DNS packet. The four-byte sdhm hash contains eight hexadecimal values, e.g., 1F7B032A. Thus, there are a total of 4,294,967,296 ( $16^8$ ) unique hashes for a four-byte hash. The maximum hash list size of 131,072,000 unique items for the TRAPP-2 system equates to 3.05% of the total number of hashes due to the 512 MB memory limit. If a system of 4,294,967,296 unique hashes is desired, then 16 GB of storage is required. The number 4,294,967,296 can be achieved by doubling the max list size of 131,072,000 approximately five more times. With a mean of approximately 16 additional CPU cycles per doubling of the hashlist, 4,294,967,296 unique hash items can be searched in additional  $5 \times 16 = 80$  CPU cycles. This assumes the hash list is sorted to cater to the binary search

algorithm. These results are encouraging for future research that will rely on larger hash list sizes.

#### ***4.5 Overall Analysis***

The results from Experiment 1 assist in understanding the results of Experiment 3. The smaller packet processing times for BitTorrent and DNS packets allows the TRAPP-2 system to capture them with a probability of packet intercept greater than 90% under the maximum network utilization of 93.7%. Both the SIP INVITE and SIP BYE packets take longer to process and, as result, are captured at a significantly lower probability of packet intercept under the same 93.7% network utilization. As a reminder, the worst-case scenario packets are selected for Experiment 3 so the probability of packet intercept represents the minimum probability of packet intercept. Higher probabilities of packet intercept can be achieved with packets that have more favorable packet characteristics. Overall, the probability of packet intercept for Wireshark is markedly lower than the TRAPP-2 system under the various network utilizations of Experiment 3. The default buffer size for Wireshark, which is used for this research, is 1 MB. This is significantly greater than the 32 KB First-In-First-Out buffer used in conjunction with the FPGA's Ethernet controller. Perhaps increasing the buffer size in Wireshark can produce more favorable results, but the fact still remains that the TRAPP-2 system's buffer is smaller and outperforms Wireshark.

Experiment 2 reveals that Wireshark outperforms the TRAPP-2 system when there is no additional network utilization added and 400 packets flood the system. It appears Wireshark is capable of handling a small flood of packets of interest more efficiently than the TRAPP-2 system. This is likely due to the large 1 MB buffer and lack of packet processing required by Wireshark.

The results from Experiment 4 reveal that doubling the hash list size increases the packet processing time only slightly (an average of 0.27%). Although the worst-case scenario DNS packet is selected for the test, the experiment essentially measures the speed of the binary search algorithm and is independent of the type of packet hash

being processed. Therefore, the results can be extended and expected of BitTorrent and SIP packet hashes as well. The binary search algorithm is chosen for simplicity. Implementing other data structures and algorithms could result in faster hash lookups.

#### **4.6 *Summary***

This chapter presents the results and analysis from the four experiments measuring packet processing time and probability of packet intercept. Statistical analysis of the data's packet processing time and probability of packet intercept is performed. An overall analysis and discussion is presented at the end. The most relevant results show that the TRAPP-2 system captures 95.56% of BitTorrent, 20.78% of SIP INVITE, 37.11% of SIP BYE, and 91.89% of DNS worst-case scenario packets of interest while under a 93.7% network utilization. Additionally, Experiment 4 reveals that each doubling of the hash list size results in a mean increase of approximately 16 CPU cycles.

## V. Conclusions

This chapter summarizes the overall goals and conclusions of the research. Section 5.1 summarizes the results and whether the goals and hypotheses are met. The significance of the research is presented in Section 5.2. Lastly, Section 5.3 provides recommendations to expand and progress the research of the TRAPP-2 system.

### 5.1 *Conclusions of Research*

#### 5.1.1 *Goal #1: Determine the packet processing times for packets of interest.*

The first goal is to determine the packet processing times for packets of interest. The TRAPP-2 system must be able to process packets as quickly as possible. Experiment 1 reveals BitTorrent and DNS packets require the fewest CPU cycles and both types of SIP packets require the most. The TRAPP-2 system can process all types of packets under 35,000 CPU cycles, thus meeting the goal and proving the hypothesis.

5.1.2 *Goal #2: Determine the probability of packet intercept under a flood of 400 packets of interest.* The second goal of this research is to determine the probability of packet intercept for a flood of 400 packets of interest. Experiment 2 reveals that the TRAPP-2 system captures 100% of BitTorrent, 86.58% of DNS, 22.00% of SIP BYE, and 19.92% of SIP INVITE packets when 400 packets are sent as fast as possible, thus meeting the research goal. However, the measured network utilizations vary significantly depending on the type of packet being sent. The TRAPP-2 system fails to capture over 50% of the SIP BYE and SIP INVITE packets, thus failing to meet the hypothesis. This type of traffic is unrealistic and is simply meant to stress the TRAPP-2 system.

5.1.3 *Goal #3: Determine the probability of packet intercept under various network utilizations.* The third goal of this research is to determine the probability of packet intercept for packets of interest under various network utilizations. Experiment 3 reveals that the TRAPP-2 system captures, with 95% confidence, 95.56% of BitTorrent, 20.78% of SIP INVITE, 37.11% of SIP BYE, and 91.89% of DNS packets

of interest under a 93.7% network utilization. The packets selected for the experiment are the worst-case scenario, so the reported probability of packet intercept is the minimum. The 93.7% network utilization is equal to 937 megabits per second. These results exceed the hypothesized values and meet the research goal.

#### *5.1.4 Goal #4: Determine how the hash list size affects packet processing time.*

The fourth goal of this research is to determine how increasing the hash list size affects the packet processing time. The original hash list size of 1,000 unique items is doubled 17 times to generate a hash list with 131,072,000 unique items. Experiment 4 reveals how each doubling of the hash list exposes the logarithmic nature of the packet processing time versus the number of hash list items. The mean packet processing time increases an average of 15.93 CPU cycles per doubling of the hash list size. This value is less than the hypothesized value of 50, thus meeting the goal and proving the hypothesis.

## **5.2 Significance of Research**

This research allows the military and government agencies to detect and track malicious BitTorrent, SIP, and DNS traffic traversing networks at gigabit speeds using large hash lists. The experiments selected measure the packet processing time and probability of packet intercept for the TRAPP-2 system under various conditions. The results and analysis conclude that the TRAPP-2 system is capable of detecting and tracking traffic of interest on a gigabit Ethernet network. This research also reveals how increasing the hash list size affects the packet processing time for DNS packets. Although DNS is the only protocol tested, the results apply equally to BitTorrent file hashes and SIP URI domains. This allows for larger lists of known illegal BitTorrent file hashes or SIP URIs of interest to be used on the TRAPP-2 system.

The TRAPP-2 system is attractive to network administrators because it is a passive solution. The FPGA design allows for quick implementation onto a local area network, assuming there is a gateway switch with a SPAN port. In addition, if the

TRAPP-2 system fails, it cannot disrupt or interfere with network traffic because it is not installed in-line with other network appliances.

For BitTorrent traffic, the TRAPP-2 system aids law enforcement in the fight against illegal file distribution. TRAPP-2 can identify the parties participating in an illegal file transfer. The TRAPP-2 system can also be used to identify the accidental or intentional disclosure of sensitive documents from military and government networks through BitTorrent file sharing programs. The system provides proof in the form of the logged packet which contains the hash of the file being transmitted and the IP addresses of the computers participating.

The proliferation of Internet phones and VoIP has made tracking persons of interest difficult. The TRAPP-2 system aids law enforcement and intelligence agencies in identifying social networks and cells of criminals, terrorists, and other people of interest using VoIP technologies. By detecting SIP URI domains of interest, maps of players and organizational hierarchies can be derived to aid investigators.

Lastly, the TRAPP-2 system aids network administrators in detecting potential data exfiltration via malicious DNS traffic. By establishing a DNS whitelist of approved domains, network administrators can use the TRAPP-2 system to identify potential abuses of DNS. The TRAPP-2 system logs the packet, and more importantly, the IP addresses of the computers communicating. This can help investigators in identifying compromised computers and the external IP addresses establishing the unauthorized communication channels.

### ***5.3 Recommendations for Future Research***

The first suggestion for future research is to expand the hardware capabilities of the ML510 FPGA. The board contains an additional PowerPC processor and gigabit Ethernet controller that are not used in this research. Additional processing, functions, and algorithm work can be offloaded to the second processor. Additionally, the second processor can be used to hash the SIP URIs and DNS domains using a better

hashing algorithm than sdbm. The second gigabit Ethernet controller can be used as a backup or out-of-band administrative Ethernet controller.

Secondly, future research can focus on using a proven hashing algorithm such as SHA-1 or the Message-Digest algorithm 5 (MD5). For this research, the sdbm hashing algorithm is selected because of its speed and simplicity. Shortfalls such as a minimal avalanche effect and potential collisions are not considered. The algorithm processing can reside on a separate dedicated processor as mentioned above.

BitTorrent, SIP, and DNS are not the only protocols that can be abused or have malicious intent. Another suggestion is to investigate the Internet Relay Chat protocol used for botnet command and control. The Hypertext Transfer Protocol is another popular tunneling protocol because of its reliance on TCP port 80, which is open in most organizations. The TRAPP-2 system can easily be adapted to other protocols that can be abused by modifying the signature detection logic.

Lastly, detecting encrypted and obfuscated network traffic is another area of future research. This research assumes that the BitTorrent, SIP, and DNS traffic is not encrypted or obfuscated. Research efforts can focus on decrypting traffic on the fly, perhaps using the other processor to accomplish this, while still maintaining the system's speed.

## *Appendix A. Experimental Data*

This Appendix contains the raw data collected for the experiments. Section A.1 contains the data from Experiment 1. Section A.2 contains the data from Experiment 2. Section A.3 contains the data from Experiment 3. Section A.4 contains the data from Experiment 4.

### *A.1 Results of Experiment 1*



Table A.1: CPU Cycle Data for Experiment 1.

Packet	Non BT/SIP/DNS			BT ON WORST			BT ON BEST			BT OFF		
	Rep. 1	Rep 2	Rep 3	Rep. 1	Rep 2	Rep 3	Rep. 1	Rep 2	Rep 3	Rep. 1	Rep 2	Rep 3
1	4985	4985	4985	2305	2205	2181	2213	2051	2049	1973	1973	1973
2	4985	4985	4985	2184	2185	2181	2052	2053	2049	1973	1973	1973
3	4985	4985	4985	2307	2321	2181	2175	2153	2049	1973	1973	1973
4	4985	4985	4985	2275	2184	2181	2143	2052	2049	1973	1973	1973
5	4985	4985	4985	2185	2279	2181	2053	2147	2049	1973	1973	1973
6	4985	4985	4985	2275	2303	2181	2143	2171	2049	1973	1973	1973
7	4985	4985	4985	2183	2185	2181	2051	2053	2049	1973	1973	1973
8	4985	4985	4985	2185	2275	2181	2053	2143	2049	1973	1973	1973
9	4985	4985	4985	2285	2183	2181	2153	2051	2049	1973	1973	1973
10	4985	4985	4985	2184	2185	2181	2052	2053	2049	1973	1973	1973
11	4985	4985	4985	2287	2285	2181	2155	2153	2049	1973	1973	1973
12	4985	4985	4985	2275	2184	2181	2143	2052	2049	1973	1973	1973
13	4985	4985	4985	2185	2287	2181	2053	2155	2049	1973	1973	1973
14	4985	4985	4985	2275	2303	2181	2143	2187	2049	1973	1973	1973
15	4985	4985	4985	2183	2185	2181	2051	2053	2049	1973	1973	1973
16	4985	4985	4985	2185	2303	2181	2053	2171	2049	1973	1973	1973
17	4985	4985	4985	2285	2183	2181	2153	2051	2049	1973	1973	1973
18	4985	4985	4985	2184	2185	2181	2052	2053	2049	1973	1973	1973
19	4985	4985	4985	2279	2305	2181	2147	2173	2049	1973	1973	1973
20	4985	4985	4985	2303	2184	2181	2171	2052	2049	1973	1973	1973
21	4985	4985	4985	2185	2307	2181	2053	2175	2049	1973	1973	1973
22	4985	4985	4985	2283	2303	2181	2151	2171	2049	1973	1973	1973
23	4985	4985	4985	2183	2185	2181	2051	2053	2049	1973	1973	1973
24	4985	4985	4985	2185	2267	2181	2053	2135	2049	1973	1973	1973
25	4985	4985	4985	2285	2183	2181	2153	2051	2049	1973	1973	1973
26	4985	4985	4985	2184	2185	2181	2052	2053	2049	1973	1973	1973
27	4985	4985	4985	2307	2277	2181	2175	2145	2049	1973	1973	1973
28	4985	4985	4985	2303	2184	2181	2171	2052	2049	1973	1973	1973
29	4985	4985	4985	2185	2271	2181	2053	2139	2049	1973	1973	1973
30	4985	4985	4985	2303	2255	2181	2171	2123	2049	1973	1973	1973
31	4985	4985	4985	2183	2185	2181	2051	2053	2049	1973	1973	1973
32	4985	4985	4985	2185	2295	2181	2053	2163	2049	1973	1973	1973
33	4985	4985	4985	2277	2183	2181	2209	2051	2049	1973	1973	1973
34	4985	4985	4985	2184	2185	2181	2052	2053	2049	1973	1973	1973
35	4985	4985	4985	2307	2337	2181	2175	2137	2049	1973	1973	1973
36	4985	4985	4985	2295	2184	2181	2151	2052	2049	1973	1973	1973
37	4985	4985	4985	2185	2271	2181	2053	2139	2049	1973	1973	1973
38	4985	4985	4985	2275	2267	2181	2143	2135	2049	1973	1973	1973
39	4985	4985	4985	2183	2185	2181	2051	2053	2049	1973	1973	1973
40	4985	4985	4985	2185	2283	2181	2053	2219	2049	1973	1973	1973
41	4985	4985	4985	2277	2183	2181	2145	2051	2049	1973	1973	1973
42	4985	4985	4985	2184	2185	2181	2052	2053	2049	1973	1973	1973
43	4985	4985	4985	2307	2232	2181	2175	2100	2049	1973	1973	1973
44	4985	4985	4985	2303	2184	2181	2171	2052	2049	1973	1973	1973
45	4985	4985	4985	2185	2287	2181	2053	2155	2049	1973	1973	1973
46	4985	4985	4985	2283	2283	2181	2151	2151	2049	1973	1973	1973
47	4985	4985	4985	2183	2185	2181	2051	2053	2049	1973	1973	1973
48	4985	4985	4985	2185	2283	2181	2053	2151	2049	1973	1973	1973
49	4985	4985	4985	2305	2183	2181	2173	2051	2049	1973	1973	1973
50	4985	4985	4985	2184	2185	2181	2052	2053	2049	1973	1973	1973

Table A.2: CPU Cycle Data for Experiment 1 Continued.

SIP INV ON LARGE WORST			SIP INV ON SMALL BEST			SIP INV OFF LARGE			SIP BYE ON LARGE WORST		
Rep. 1	Rep 2	Rep 3	Rep. 1	Rep 2	Rep 3	Rep. 1	Rep 2	Rep 3	Rep. 1	Rep 2	Rep 3
33240	32776	32802	14823	14735	14761	31092	31092	31092	25615	25347	25388
33191	32809	32815	14929	14815	14797	31092	31092	31092	25448	25374	25347
33257	32896	32872	14769	14822	14812	31092	31092	31092	25579	25347	25347
33341	32779	32800	14859	14769	14734	31092	31092	31092	25496	25347	25388
33071	32856	32830	14863	14783	14804	31092	31092	31092	25529	25347	25347
33292	32857	32863	14757	14758	14776	31092	31092	31092	25553	25370	25388
33041	32780	32820	14970	14774	14774	31092	31092	31092	25545	25347	25383
33034	32847	32856	14848	14782	14806	31092	31092	31092	25613	25380	25347
32938	32832	32812	14734	14761	14755	31092	31092	31092	25629	25347	25392
33046	32805	32801	14825	14807	14786	31092	31092	31092	25553	25380	25376
33620	32825	32858	14858	14783	14822	31092	31092	31092	25629	25376	25347
33527	32800	32779	14857	14734	14769	31092	31092	31092	25649	25384	25376
34028	32862	32862	14857	14820	14824	31092	31092	31092	25448	25347	25388
34629	32907	32897	14838	14761	14734	31092	31092	31092	25468	25374	25368
34350	32816	32812	14836	14760	14766	31092	31092	31092	25496	25368	25347
35319	35268	35304	14768	14816	14808	31092	31092	31092	25462	25370	25380
36279	36194	36194	14770	14735	14735	31092	31092	31092	25615	25376	25396
34572	34571	34597	14805	14851	14867	31092	31092	31092	25735	25374	25347
34632	34647	34665	14797	14808	14822	31092	31092	31092	25735	25368	25384
34646	34596	34572	14867	14734	14734	31092	31092	31092	25551	25347	25347
34624	34632	34634	15022	14810	14783	31092	31092	31092	26238	25384	25388
34537	34706	34666	14928	14734	14775	31092	31092	31092	26540	25347	25388
34590	34600	34596	14862	14734	14734	31092	31092	31092	26528	25368	25384
34597	34655	34627	15103	14782	14802	31092	31092	31092	26276	25347	25347
34647	34590	34522	15726	14771	14735	31092	31092	31092	26935	26311	26307
34596	34567	34597	15882	14823	14819	31092	31092	31092	28288	28600	28534
34658	34641	34641	15662	14835	14835	31092	31092	31092	26566	26508	26534
34574	34534	34490	15735	14769	14734	31092	31092	31092	26566	26510	26508
34438	34620	34618	15181	16279	16155	31092	31092	31092	26608	26566	26540
34653	34670	34696	17071	16833	16879	31092	31092	31092	26582	26550	26508
34594	34596	34596	15828	15752	15726	31092	31092	31092	26558	26576	26576
34597	34601	34627	15803	15828	15832	31092	31092	31092	26568	26576	26510
34605	34390	34390	15726	15728	15728	31092	31092	31092	26546	26602	26566
34582	34597	34597	15804	15831	15873	31092	31092	31092	26540	26604	26534
34510	34397	34365	15688	15799	15799	31092	31092	31092	26540	26576	26470
35495	35555	35679	15831	15753	15729	31092	31092	31092	26580	26508	26600
34452	35158	35202	15847	15812	15840	31092	31092	31092	26542	26508	26552
35971	35872	35890	15753	15753	15727	31092	31092	31092	26566	26552	26508
34590	34596	34596	15828	15752	15726	31092	31092	31092	26500	26534	26508
34597	34627	34603	15727	15802	15830	31092	31092	31092	26608	26508	26566
34665	34590	34594	15752	15752	15726	31092	31092	31092	26590	26534	26534
34596	34597	34597	15802	15831	15805	31092	31092	31092	26542	26550	26508
34632	34665	34639	15688	15799	15835	31092	31092	31092	26502	26508	26550
34670	34610	34596	15831	15727	15727	31092	31092	31092	26526	26536	26534
34532	34620	34632	15825	15844	15802	31092	31092	31092	26618	26550	26508
34627	34676	34670	15753	15759	15769	31092	31092	31092	26634	26444	26486
34566	34598	34600	15828	15726	15760	31092	31092	31092	26608	26508	26508
34507	34563	34495	15727	15828	15828	31092	31092	31092	26560	26508	26534
34649	34642	34664	15728	15726	15752	31092	31092	31092	26478	26602	26508
34572	34597	34597	15738	15807	15807	31092	31092	31092	26636	26594	26508

Table A.3: CPU Cycle Data for Experiment 1 Continued.

SIP BYE ON SMALL BEST			DNS ON LARGE WORST			DNS OFF SMALL			DNS OFF LARGE		
Rep. 1	Rep 2	Rep 3	Rep. 1	Rep 2	Rep 3	Rep. 1	Rep 2	Rep 3	Rep. 1	Rep 2	Rep 3
8522	8623	8623	5172	5172	5172	1635	1635	1635	5504	5566	5492
8624	8522	8522	5172	5172	5172	1757	1757	1635	5606	5604	5492
8528	8658	8662	5172	5172	5172	1635	1761	1635	5504	5690	5492
8638	8522	8522	5172	5172	5172	1729	1635	1635	5626	5560	5492
8522	8623	8623	5172	5172	5172	1733	1759	1635	5504	5582	5492
8644	8522	8522	5172	5172	5172	1635	1635	1635	5626	5560	5492
8544	8623	8658	5172	5172	5172	1759	1635	1635	5504	5698	5492
8644	8522	8522	5172	5172	5172	1635	1729	1635	5598	5504	5492
8522	8623	8623	5172	5172	5172	1635	1635	1635	5504	5582	5492
8616	8522	8522	5172	5172	5172	1757	1729	1635	5626	5504	5492
8560	8623	8623	5172	5172	5172	1635	1761	1635	5504	5582	5492
8624	8522	8522	5172	5172	5172	1729	1635	1635	5598	5504	5492
8546	8623	8658	5172	5172	5172	1741	1739	1635	5504	5582	5492
8644	8522	8522	5172	5172	5172	1635	1635	1635	5626	5504	5492
8522	8623	8623	5172	5172	5172	1823	1635	1635	5504	5582	5492
8644	8522	8522	5172	5172	5172	1635	1729	1635	5626	5556	5492
8522	8623	8648	5172	5172	5172	1635	1635	1635	5504	5582	5492
8616	8556	8522	5172	5172	5172	1737	1757	1635	5598	5560	5492
8522	8623	8623	5172	5172	5172	1635	1733	1635	5504	5582	5492
8616	8522	8522	5172	5172	5172	1737	1635	1635	5626	5571	5492
8522	8623	8623	5172	5172	5172	1761	1731	1635	5504	5606	5492
8616	8522	8544	5172	5172	5172	1635	1635	1635	5626	5609	5492
8522	8623	8623	5172	5172	5172	1759	1635	1635	5504	5778	5492
8664	8522	8538	5172	5172	5172	1635	1757	1635	5658	5504	5492
8546	8623	8644	5172	5172	5172	1635	1635	1635	5504	5663	5492
8632	8522	8522	5172	5172	5172	1729	1729	1635	5626	5504	5492
8522	8678	8639	5172	5172	5172	1635	1741	1635	5504	5582	5492
8660	8522	8562	5172	5172	5172	1757	1635	1635	5598	5608	5492
8522	8623	8623	5172	5172	5172	1733	1739	1635	5504	5582	5492
8644	8522	8522	5172	5172	5172	1635	1635	1635	5598	5504	5492
8522	8650	8650	5172	5172	5172	1731	1635	1635	5504	5582	5492
8644	8522	8552	5172	5172	5172	1635	1757	1635	5598	5504	5492
8522	8650	8623	5172	5172	5172	1635	1635	1635	5504	5582	5492
8616	8522	8522	5172	5172	5172	1737	1729	1635	5626	5504	5492
8522	8623	8623	5172	5172	5172	1635	1733	1635	5504	5582	5492
8652	8522	8522	5172	5172	5172	1757	1635	1635	5606	5504	5492
8522	8623	8623	5172	5172	5172	1733	1739	1635	5504	5582	5492
8644	8522	8562	5172	5172	5172	1635	1635	1635	5598	5504	5492
8522	8623	8623	5172	5172	5172	1731	1635	1635	5504	5582	5492
8608	8522	8548	5172	5172	5172	1635	1729	1635	5626	5504	5492
8522	8656	8648	5172	5172	5172	1635	1635	1635	5533	5582	5492
8624	8522	8522	5172	5172	5172	1737	1737	1635	5598	5504	5492
8522	8662	8623	5172	5172	5172	1635	1733	1635	5504	5582	5492
8624	8522	8522	5172	5172	5172	1757	1635	1635	5626	5504	5492
8522	8623	8623	5172	5172	5172	1801	1759	1635	5504	5582	5492
8624	8554	8522	5172	5172	5172	1635	1635	1635	5598	5504	5492
8522	8662	8662	5172	5172	5172	1759	1635	1635	5504	5582	5492
8624	8522	8522	5172	5172	5172	1635	1729	1635	5598	5504	5492
8522	8711	8623	5172	5172	5172	1635	1635	1635	5541	5582	5492
8571	8554	8522	5172	5172	5172	1737	1757	1635	5626	5541	5492

## A.2 Results of Experiment 2

Table A.4: Packets Captured for Experiment 2.

Packet Type	Packets Captured (Events)			Total Packets Captured	Packets Sent (Trials)
	Rep. 1	Rep. 2	Rep. 3		
BT ON WORST	400	400	400	1200	1200
Wireshark	400	400	400	1200	1200
				Average	
Load (Mbps)	156.664	153.579	153.449	154.564	
% of Max	15.67%	15.36%	15.34%	15.46%	
SIP INVITE ON LARGE WORST	79	80	80	239	1200
Wireshark	400	400	400	1200	1200
				Average	
Load (Mbps)	994.402	997.901	992.099	994.801	
% of Max	99.44%	99.79%	99.21%	99.48%	
SIP BYE ON LARGE WORST	88	88	88	264	1200
Wireshark	400	400	400	1200	1200
				Average	
Load (Mbps)	944.62	952.483	945.451	947.518	
% of Max	94.46%	95.25%	94.55%	94.75%	
DNS OFF LARGE	343	348	348	1039	1200
Wireshark	400	400	400	1200	1200
				Average	
Load (Mbps)	242.339	243.776	228.405	238.173	
% of Max	24.23%	24.38%	22.84%	23.82%	

### A.3 Results of Experiment 3

Table A.5: Packets Captured for Experiment 3, Utilization 1 ( $\approx 20.4\%$ ).

Utilization 1 ( $\approx 20.4\%$ )					
Packet Type	Packets Captured (Events)			Total Packets Captured	Packets Sent (Trials)
	Rep. 1	Rep. 2	Rep. 3		
BT ON WORST	300	300	300	900	900
Wireshark	286	288	289	863	900
				Average	
Utilization (Mbps)	204	204	204	204	
% of Max	20.40%	20.40%	20.40%	20.40%	
SIP INVITE ON LARGE WORST	300	300	300	900	900
Wireshark	290	284	290	864	900
				Average	
Utilization (Mbps)	204	204	204	204	
% of Max	20.40%	20.40%	20.40%	20.40%	
SIP BYE ON LARGE WORST	300	300	300	900	900
Wireshark	296	285	286	867	900
				Average	
Utilization (Mbps)	204	204	204	204	
% of Max	20.40%	20.40%	20.40%	20.40%	
DNS OFF LARGE	300	300	300	900	900
Wireshark	295	284	282	861	900
				Average	
Utilization (Mbps)	204	204	204	204	
% of Max	20.40%	20.40%	20.40%	20.40%	

Table A.6: Packets Captured for Experiment 3, Utilization 2 ( $\approx 30.1\%$ ).

Utilization 2 ( $\approx 30.10\%$ )					
Packet Type	Packets Captured (Events)			Total Packets Captured	Packets Sent (Trials)
	Rep. 1	Rep. 2	Rep. 3		
BT ON WORST	290	293	293	876	900
Wireshark	210	216	219	645	900
				Average	
Utilization (Mbps)	301	301	301	301	
% of Max	30.10%	30.10%	30.10%	30.10%	
SIP INVITE ON LARGE WORST	136	128	119	383	900
Wireshark	209	208	201	618	900
				Average	
Utilization (Mbps)	301	301	301	301	
% of Max	30.10%	30.10%	30.10%	30.10%	
SIP BYE ON LARGE WORST	192	190	189	571	900
Wireshark	187	216	207	610	900
				Average	
Utilization (Mbps)	301	301	301	301	
% of Max	30.10%	30.10%	30.10%	30.10%	
DNS OFF LARGE	294	294	293	881	900
Wireshark	182	184	223	589	900
				Average	
Utilization (Mbps)	301	301	301	301	
% of Max	30.10%	30.10%	30.10%	30.10%	

Table A.7: Packets Captured for Experiment 3, Utilization 3 ( $\approx 40.8\%$ ).

Utilization 3 ( $\approx 40.8\%$ )					
Packet Type	Packets Captured (Events)			Total Packets Captured	Packets Sent (Trials)
	Rep. 1	Rep. 2	Rep. 3		
BT ON WORST	290	297	283	870	900
Wireshark	124	137	110	371	900
				Average	
Utilization (Mbps)	408	408	408	408	
% of Max	40.80%	40.80%	40.80%	40.80%	
SIP INVITE ON LARGE WORST	88	98	97	283	900
Wireshark	152	136	114	402	900
				Average	
Utilization (Mbps)	408	408	408	408	
% of Max	40.80%	40.80%	40.80%	40.80%	
SIP BYE ON LARGE WORST	156	158	153	467	900
Wireshark	123	131	124	378	900
				Average	
Utilization (Mbps)	408	408	408	408	
% of Max	40.80%	40.80%	40.80%	40.80%	
DNS OFF LARGE	287	292	289	868	900
Wireshark	138	140	127	405	900
				Average	
Utilization (Mbps)	408	408	408	408	
% of Max	40.80%	40.80%	40.80%	40.80%	

Table A.8: Packets Captured for Experiment 3, Utilization 4 ( $\approx 49.8\%$ ).

Utilization 4 ( $\approx 49.8\%$ )					
Packet Type	Packets Captured (Events)			Total Packets Captured	Packets Sent (Trials)
	Rep. 1	Rep. 2	Rep. 3		
BT ON WORST	292	292	290	874	900
Wireshark	106	111	119	336	900
				Average	
Utilization (Mbps)	498	498	498	498	
% of Max	49.80%	49.80%	49.80%	49.80%	
SIP INVITE ON LARGE WORST	91	77	78	246	900
Wireshark	89	112	112	313	900
				Average	
Utilization (Mbps)	498	498	498	498	
% of Max	49.80%	49.80%	49.80%	49.80%	
SIP BYE ON LARGE WORST	145	143	126	414	900
Wireshark	91	91	95	277	900
				Average	
Utilization (Mbps)	498	498	498	498	
% of Max	49.80%	49.80%	49.80%	49.80%	
DNS OFF LARGE	292	287	286	865	900
Wireshark	129	95	96	320	900
				Average	
Utilization (Mbps)	498	498	498	498	
% of Max	49.80%	49.80%	49.80%	49.80%	

Table A.9: Packets Captured for Experiment 3, Utilization 5 ( $\approx 60.2\%$ ).

Utilization 5 ( $\approx 60.2\%$ )					
Packet Type	Packets Captured (Events)			Total Packets Captured	Packets Sent (Trials)
	Rep. 1	Rep. 2	Rep. 3		
BT ON WORST	285	286	291	862	900
Wireshark	74	80	76	230	900
				Average	
Utilization (Mbps)	602	602	602	602	
% of Max	60.20%	60.20%	60.20%	60.20%	
SIP INVITE ON LARGE WORST	97	79	74	250	900
Wireshark	77	81	75	233	900
				Average	
Utilization (Mbps)	602	602	602	602	
% of Max	60.20%	60.20%	60.20%	60.20%	
SIP BYE ON LARGE WORST	166	144	155	465	900
Wireshark	97	103	80	280	900
				Average	
Utilization (Mbps)	602	602	602	602	
% of Max	60.20%	60.20%	60.20%	60.20%	
DNS OFF LARGE	266	278	275	819	900
Wireshark	89	95	99	283	900
				Average	
Utilization (Mbps)	602	602	602	602	
% of Max	60.20%	60.20%	60.20%	60.20%	

Table A.10: Packets Captured for Experiment 3, Utilization 6 ( $\approx 71.4\%$ ).

Utilization 6 ( $\approx 71.4\%$ )					
Packet Type	Packets Captured (Events)		Total Packets Captured		Packets Sent (Trials)
	Rep. 1	Rep. 2	Rep. 3		
BT ON WORST	285	278	288	851	900
Wireshark	77	76	59	212	900
				Average	
Utilization (Mbps)	714	714	714	714	
% of Max	71.40%	71.40%	71.40%	71.40%	
SIP INVITE ON LARGE WORST	75	76	76	227	900
Wireshark	67	63	73	203	900
				Average	
Utilization (Mbps)	714	714	714	714	
% of Max	71.40%	71.40%	71.40%	71.40%	
SIP BYE ON LARGE WORST	146	136	142	424	900
Wireshark	85	71	74	230	900
				Average	
Utilization (Mbps)	714	714	714	714	
% of Max	71.40%	71.40%	71.40%	71.40%	
DNS OFF LARGE	279	270	279	828	900
Wireshark	71	69	66	206	900
				Average	
Utilization (Mbps)	714	714	714	714	
% of Max	71.40%	71.40%	71.40%	71.40%	

Table A.11: Packets Captured for Experiment 3, Utilization 7 ( $\approx 81.8\%$ ).

Utilization 7 ( $\approx 81.8\%$ )					
Packet Type	Packets Captured (Events)		Total Packets Captured		Packets Sent (Trials)
	Rep. 1	Rep. 2	Rep. 3		
BT ON WORST	284	286	292	862	900
Wireshark	72	59	57	188	900
				Average	
Utilization (Mbps)	818	818	818	818	
% of Max	81.80%	81.80%	81.80%	81.80%	
SIP INVITE ON LARGE WORST	63	64	82	209	900
Wireshark	56	77	65	198	900
				Average	
Utilization (Mbps)	818	818	818	818	
% of Max	81.80%	81.80%	81.80%	81.80%	
SIP BYE ON LARGE WORST	152	144	113	409	900
Wireshark	73	56	50	179	900
				Average	
Utilization (Mbps)	818	818	818	818	
% of Max	81.80%	81.80%	81.80%	81.80%	
DNS OFF LARGE	268	265	269	802	900
Wireshark	75	79	71	225	900
				Average	
Utilization (Mbps)	818	818	818	818	
% of Max	81.80%	81.80%	81.80%	81.80%	



Table A.12: Packets Captured for Experiment 3, Utilization 8 ( $\approx 93.7\%$ ).

Utilization 8 (Max $\approx 93.7\%$ )					
Packet Type	Packets Captured (Events)			Total Packets Captured	Packets Sent (Trials)
	Rep. 1	Rep. 2	Rep. 3		
BT ON WORST	287	289	284	860	900
Wireshark	54	48	53	155	900
				Average	
Utilization (Mbps)	937	937	937	937	
% of Max	93.70%	93.70%	93.70%	93.70%	
SIP INVITE ON LARGE WORST	72	58	57	187	900
Wireshark	53	53	41	147	900
				Average	
Utilization (Mbps)	937	937	937	937	
% of Max	93.70%	93.70%	93.70%	93.70%	
SIP BYE ON LARGE WORST	117	106	111	334	900
Wireshark	53	46	44	143	900
				Average	
Utilization (Mbps)	937	937	937	937	
% of Max	93.70%	93.70%	93.70%	93.70%	
DNS OFF LARGE	273	275	279	827	900
Wireshark	53	59	50	162	900
				Average	
Utilization (Mbps)	937	937	937	937	
% of Max	93.70%	93.70%	93.70%	93.70%	

## A.4 Results of Experiment 4

Table A.13: CPU Cycle Data for Experiment 4.

List Size	2,000				4,000				8,000				16,000		
Packet	Rep 1	Rep 2	Rep 3		Rep 1	Rep 2	Rep 3		Rep 1	Rep 2	Rep 3		Rep 1	Rep 2	Rep 3
1	5646	5759	5636		5662	5662	5650		5678	5678	5666		5694	5694	5682
2	5748	5646	5636		5764	5738	5650		5754	5743	5666		5770	5754	5682
3	5646	5722	5636		5662	5662	5650		5678	5678	5666		5743	5742	5682
4	5748	5729	5636		5764	5775	5650		5743	5754	5666		5759	5770	5682
5	5646	5722	5636		5662	5662	5650		5678	5678	5666		5694	5694	5682
6	5768	5646	5636		5764	5738	5650		5754	5791	5666		5743	5770	5682
7	5646	5722	5636		5662	5662	5650		5678	5678	5666		5694	5694	5682
8	5768	5646	5636		5764	5738	5650		5738	5738	5666		5770	5770	5682
9	5646	5722	5636		5662	5675	5650		5678	5678	5666		5694	5694	5682
10	5744	5646	5636		5764	5790	5650		5738	5756	5666		5743	5759	5682
11	5646	5722	5636		5662	5662	5650		5717	5678	5666		5694	5694	5682
12	5740	5646	5636		5764	5738	5650		5754	5754	5666		5770	5759	5682
13	5679	5722	5636		5662	5662	5650		5678	5678	5666		5694	5694	5682
14	5803	5646	5636		5764	5738	5650		5771	5754	5666		5759	5824	5682
15	5691	5722	5636		5662	5662	5650		5678	5678	5666		5694	5694	5682
16	5768	5698	5636		5776	5738	5650		5754	5779	5666		5743	5811	5682
17	5646	5735	5636		5662	5714	5650		5678	5678	5666		5694	5694	5682
18	5768	5646	5636		5764	5738	5650		5754	5816	5666		5823	5840	5682
19	5646	5722	5636		5662	5662	5650		5678	5678	5666		5694	5745	5682
20	5740	5646	5636		5764	5738	5650		5754	5754	5666		5743	5807	5682
21	5646	5722	5636		5662	5662	5650		5678	5742	5666		5694	5694	5682
22	5740	5646	5636		5764	5738	5650		5754	5878	5666		5754	5770	5682
23	5646	5780	5636		5662	5662	5650		5678	5678	5666		5694	5746	5682
24	5748	5646	5636		5764	5738	5650		5743	5775	5666		5759	5902	5682
25	5646	5930	5636		5662	5662	5650		5678	5678	5666		5694	5787	5682
26	5748	5646	5636		5764	5738	5650		5743	5846	5666		5770	5796	5682
27	5646	5722	5636		5662	5890	5650		5678	5726	5666		5694	5726	5682
28	5768	5846	5636		5764	5935	5650		5793	5754	5666		5743	6023	5682
29	5646	5814	5636		5699	5879	5650		5678	5698	5666		5694	5694	5682
30	5748	5698	5636		5764	5856	5650		5818	6006	5666		5770	5946	5682
31	5646	5706	5636		5662	5775	5650		5678	5678	5666		5694	5694	5682
32	5740	5790	5636		5764	5738	5650		5754	5806	5666		5827	5770	5682
33	5646	6090	5636		5662	5662	5650		5678	5823	5666		5694	5694	5682
34	5768	5646	5636		5764	5894	5650		5727	5966	5666		5809	5770	5682
35	5646	5722	5636		5693	5662	5650		5678	5678	5666		5694	5775	5682
36	5768	5646	5636		5764	5738	5650		5754	5754	5666		5759	5770	5682
37	5646	5722	5636		5662	5662	5650		5678	5678	5666		5694	5735	5682
38	5773	5646	5636		5764	5799	5650		5754	5754	5666		5759	5819	5682
39	5646	5792	5636		5662	5714	5650		5678	5678	5666		5694	5694	5682
40	5748	5646	5636		5764	5738	5650		5743	5754	5666		5816	5770	5682
41	5646	5722	5636		5662	5662	5650		5678	5678	5666		5694	5694	5682
42	5768	5646	5636		5801	5738	5650		5743	5754	5666		5770	5783	5682
43	5646	5748	5636		5662	5675	5650		5678	5678	5666		5694	5694	5682
44	5768	5703	5636		5764	5738	5650		5727	5754	5666		5770	5770	5682
45	5646	5722	5636		5662	5679	5650		5678	5678	5666		5694	5694	5682
46	5748	5646	5636		5784	5738	5650		5743	5754	5666		5759	5780	5682
47	5646	5722	5636		5662	5662	5650		5678	5678	5666		5694	5731	5682
48	5740	5646	5636		5764	5738	5650		5754	5754	5666		5743	5770	5682
49	5646	5722	5636		5705	5662	5650		5678	5678	5666		5694	5694	5682
50	5740	5646	5636		5764	5738	5650		5777	5754	5666		5770	5770	5682

Table A.14: CPU Cycle Data for Experiment 4 Continued.

32,000			64,000			128,000			256,000		
Rep 1	Rep 2	Rep 3	Rep 1	Rep 2	Rep 3	Rep 1	Rep 2	Rep 3	Rep 1	Rep 2	Rep 3
5710	5710	5698	5775	5726	5714	5742	5783	5738	5766	5766	5754
5786	5775	5698	5802	5791	5714	5815	5814	5738	5835	5835	5754
5710	5758	5698	5726	5741	5717	5742	5750	5738	5766	5766	5754
5775	5775	5698	5819	5863	5718	5826	5814	5738	5869	5887	5754
5710	5710	5698	5726	5726	5714	5742	5750	5738	5766	5823	5754
5775	5775	5698	5791	5802	5714	5815	5830	5738	5835	5846	5754
5710	5710	5698	5733	5726	5714	5742	5750	5738	5766	5766	5754
5828	5775	5698	5791	5802	5714	5803	5819	5738	5819	5835	5754
5710	5710	5698	5726	5726	5714	5742	5750	5738	5766	5766	5754
5759	5828	5698	5835	5837	5714	5826	5817	5738	5846	5846	5754
5733	5710	5698	5726	5726	5714	5742	5802	5738	5811	5766	5754
5794	5786	5698	5845	5791	5714	5815	5830	5738	5835	5835	5754
5710	5710	5698	5726	5726	5714	5742	5750	5738	5766	5766	5754
5791	5775	5698	5791	5791	5714	5826	5914	5738	5846	5835	5754
5710	5710	5698	5726	5726	5714	5742	5818	5738	5766	5766	5754
5775	5775	5698	5791	5791	5714	5815	5904	5738	5835	5846	5754
5710	5810	5698	5726	5726	5714	5742	5839	5738	5766	5801	5754
5786	5795	5698	5791	5791	5714	5826	5927	5738	5819	5835	5754
5710	5710	5698	5726	5778	5714	5742	5854	5738	5766	5818	5754
5786	5823	5698	5791	5811	5714	5815	5830	5738	5846	5835	5754
5710	5710	5698	5726	5882	5714	5742	6046	5738	5766	5934	5754
5775	5796	5698	5791	5797	5714	5826	5922	5738	5846	5914	5754
5710	5762	5698	5726	5726	5714	5742	5834	5738	5766	5890	5754
5835	5882	5698	5802	5886	5714	5815	5856	5738	5835	5846	5754
5710	5810	5698	5726	5890	5714	5742	5818	5738	5823	5905	5757
5775	5920	5698	5791	5855	5714	5826	5962	5738	5835	6179	5754
5710	5846	5698	5726	5726	5714	5742	5750	5738	5766	5766	5754
5775	5796	5698	5775	6202	5714	5815	5882	5738	5819	6062	5754
5717	5882	5698	5773	5778	5714	5742	5750	5738	5766	5818	5754
5775	5972	5698	5802	5802	5714	5826	5839	5738	5846	5995	5754
5710	5710	5698	5726	5990	5714	5742	5750	5738	5766	5862	5754
5811	5850	5698	5791	5791	5714	5826	5830	5738	5846	5835	5754
5710	5710	5698	5726	5726	5714	5742	5750	5738	5766	5818	5754
5775	5786	5698	5791	5825	5714	5826	5845	5738	5846	5846	5754
5710	5710	5698	5726	5726	5714	5742	5799	5738	5766	5766	5754
5786	5786	5698	5786	5802	5714	5815	5830	5738	5819	5846	5754
5710	5710	5698	5726	5726	5714	5742	5750	5738	5766	5766	5754
5775	5786	5698	5786	5802	5714	5826	5830	5738	5830	5846	5754
5710	5749	5698	5726	5726	5714	5742	5783	5738	5766	5766	5754
5786	5786	5698	5802	5802	5714	5826	5830	5738	5846	5883	5754
5710	5710	5698	5726	5726	5714	5742	5750	5738	5803	5766	5754
5823	5786	5698	5806	5802	5714	5866	5872	5738	5846	5846	5754
5710	5710	5698	5726	5765	5714	5742	5750	5738	5766	5766	5754
5786	5786	5698	5791	5802	5714	5826	5830	5738	5846	5846	5754
5710	5710	5698	5726	5726	5714	5742	5750	5738	5766	5807	5754
5811	5786	5698	5802	5802	5714	5841	5830	5738	5819	5846	5754
5710	5710	5698	5726	5726	5714	5742	5750	5738	5766	5766	5754
5775	5786	5698	5802	5802	5714	5826	5830	5738	5899	5846	5754
5710	5710	5698	5726	5749	5714	5742	5750	5738	5766	5766	5754
5786	5786	5698	5802	5802	5714	5857	5830	5738	5846	5846	5754

Table A.15: CPU Cycle Data for Experiment 4 Continued.

512,000			1,024,000			2,048,000			4,096,000		
Rep 1	Rep 2	Rep 3	Rep 1	Rep 2	Rep 3	Rep 1	Rep 2	Rep 3	Rep 1	Rep 2	Rep 3
5782	5782	5770	5798	5798	5786	5851	5814	5802	5834	5834	5822
5847	5851	5770	5878	5867	5786	5894	5954	5802	5914	5909	5822
5782	5782	5770	5798	5798	5786	5855	5814	5802	5834	5834	5822
5847	5914	5770	5867	5919	5786	5883	5878	5802	5903	5951	5822
5782	5782	5770	5798	5798	5786	5814	5814	5802	5834	5834	5822
5847	5862	5770	5867	5878	5786	5883	5878	5802	5903	5887	5822
5782	5782	5770	5798	5798	5786	5814	5814	5802	5834	5834	5822
5831	5885	5770	5851	5878	5786	5867	5894	5802	5887	5914	5822
5782	5782	5770	5798	5798	5786	5814	5814	5802	5834	6534	5822
5893	5862	5770	5878	5867	5786	5894	5883	5802	5914	5941	5822
5782	5782	5770	5798	5798	5786	5814	5814	5802	5834	5882	5822
5847	5851	5770	5878	5878	5786	5883	5883	5802	5903	5898	5822
5782	5782	5770	5798	5798	5786	5814	5814	5802	5834	5834	5822
5847	5851	5770	5867	5867	5786	5894	5935	5802	5914	5966	5822
5782	5782	5770	5798	5798	5786	5814	5814	5802	5834	5834	5822
5847	5851	5770	5867	5919	5786	5894	5950	5802	5914	5914	5822
5782	5782	5770	5798	5798	5786	5814	5814	5802	5834	5902	5822
5842	5862	5770	5867	5867	5786	5878	6033	5802	5887	6022	5822
5782	5886	5770	5798	5854	5786	5814	6414	5802	5834	5938	5822
5858	5885	5770	5867	5878	5786	5929	5883	5802	5914	5914	5822
5782	5834	5770	5798	6462	5786	5814	5866	5802	5834	5886	5822
5858	5919	5770	5867	5919	5786	5894	5894	5802	5914	6055	5822
5782	6315	5770	5798	5798	5786	5814	5942	5802	5834	5918	5822
5847	5914	5770	5878	5930	5786	5883	5894	5802	5903	5940	5822
5782	5874	5770	5798	5902	5786	5814	5886	5802	5834	5902	5822
5847	5851	5770	5867	5867	5786	5894	5972	5802	5914	6010	5822
5782	5782	5770	5798	5798	5786	5814	5962	5802	5834	5834	5822
5847	6078	5770	5851	6282	5786	5883	6018	5802	5887	5966	5822
5782	5843	5770	5798	5850	5786	5814	5814	5802	5834	5834	5822
5858	6011	5770	5927	5878	5786	5894	6047	5802	5914	5914	5822
5782	5878	5770	5798	5850	5786	5814	5914	5802	5834	5834	5822
5858	5862	5770	5878	5878	5786	5894	5883	5802	5914	5951	5822
5782	5834	5770	5798	5798	5786	5814	5866	5802	5834	5834	5822
5842	5911	5770	5867	5878	5786	5894	5894	5802	5914	5914	5822
5782	5782	5770	5798	5798	5786	5814	5814	5802	5834	5834	5822
5842	5899	5770	5862	5931	5786	5919	5894	5802	5903	5914	5822
5782	5782	5770	5811	5798	5786	5814	5814	5802	5834	5834	5822
5858	5916	5770	5862	5878	5786	5894	5894	5802	5914	5967	5822
5782	5782	5770	5798	5798	5786	5814	5814	5802	5834	5863	5822
5892	5862	5770	5896	5878	5786	5894	5894	5802	5914	5914	5822
5782	5782	5770	5798	5798	5786	5814	5814	5802	5834	5834	5822
5858	5862	5770	5878	5878	5786	5904	5894	5802	5914	5914	5822
5782	5782	5770	5798	5798	5786	5814	5814	5802	5834	5834	5822
5858	5862	5770	5867	5878	5786	5894	5894	5802	5914	5914	5822
5782	5782	5770	5798	5798	5786	5814	5814	5802	5834	5834	5822
5858	5862	5770	5878	5928	5786	5883	5894	5802	5903	5914	5822
5782	5782	5770	5798	5798	5786	5814	5814	5802	5865	5881	5822
5858	5862	5770	5878	5878	5786	5883	5894	5802	5903	5914	5822
5782	5782	5770	5798	5798	5786	5814	5814	5802	5834	5834	5822
5847	5862	5770	5878	5878	5786	5883	5894	5802	5914	5914	5822

Table A.16: CPU Cycle Data for Experiment 4 Continued.

8,192,000			16,384,000			32,768,000			65,536,000		
Rep 1	Rep 2	Rep 3	Rep 1	Rep 2	Rep 3	Rep 1	Rep 2	Rep 3	Rep 1	Rep 2	Rep 3
5850	5902	5839	5866	5866	5854	5882	6746	5870	5902	5954	5890
5930	5919	5838	5946	5935	5854	6003	5951	5870	5971	5974	5890
5850	5850	5838	5866	5866	5854	5882	5882	5870	5902	5902	5890
5919	5919	5838	5935	6679	5854	5951	5951	5870	5982	6022	5890
5850	5881	5838	5866	5866	5854	5882	5919	5870	5902	5902	5890
5919	6643	5838	5935	5935	5854	5951	5951	5870	5971	6124	5890
5850	5850	5838	5866	5866	5854	5882	5891	5870	5902	5902	5890
5903	5932	5838	5919	5946	5854	5935	6004	5870	5971	5974	5890
5850	5930	5838	5866	5918	5854	5882	5938	5870	5902	5954	5890
5930	5914	5838	5946	5943	5854	5962	5946	5870	5971	5963	5890
5850	5902	5838	5866	5918	5854	5882	5990	5870	5902	6120	5890
5919	5914	5838	5961	5930	5854	5951	5946	5870	5971	5963	5890
5850	5850	5838	5866	5871	5854	5882	5882	5870	5902	5902	5890
5930	5982	5838	5946	5998	5854	5962	6014	5870	5982	6000	5890
5850	5850	5838	5915	5866	5854	5882	5882	5870	5902	5970	5890
5930	5987	5838	5946	5946	5854	5962	5962	5870	5982	6027	5890
5850	5918	5838	5866	5934	5854	5882	5954	5870	5902	5902	5890
5963	6038	5838	5935	6054	5854	5999	6074	5870	5971	6092	5980
5850	5954	5838	5911	5970	5854	5882	5986	5870	5902	5902	5890
5988	5993	5838	5935	5946	5854	5951	5962	5870	5982	6158	5890
5850	5902	5838	5866	5918	5854	5882	5934	5870	5902	5902	5890
5952	6022	5838	5946	6106	5854	5962	6080	5870	5982	6004	5890
5850	5997	5838	5866	6014	5854	5921	5966	5870	5902	5958	5890
5919	5956	5838	5935	5972	5854	5951	5988	5870	5971	5974	5890
5850	5918	5838	5866	5934	5854	5882	5954	5870	5902	5902	5890
5930	6026	5838	5984	6042	5854	5962	6062	5870	5971	5974	5890
5850	5850	5838	5866	5866	5854	5882	5882	5870	5902	5902	5890
5930	5982	5838	5919	5998	5854	6002	6014	5870	5971	6009	5890
5850	5850	5838	5866	5903	5854	5882	5882	5870	5959	5902	5890
5930	5930	5838	5983	5946	5854	5962	5962	5870	5993	5974	5890
5850	5887	5838	5866	5866	5854	5882	5931	5870	5902	5902	5890
5930	5930	5838	5946	5946	5854	5962	5962	5870	5971	5974	5890
5850	5850	5838	5866	5866	5854	5882	5882	5870	5902	5902	5890
5930	5930	5838	5946	5946	5854	5962	6011	5870	5971	5974	5890
5850	5850	5838	5866	5866	5854	5882	5882	5870	5902	5902	5890
5919	5930	5838	5935	5946	5854	5951	5962	5870	5982	5974	5890
5850	5850	5838	5866	5866	5854	5882	5882	5870	5902	5902	5890
5930	5967	5838	5930	5946	5854	5983	5962	5870	5982	5974	5890
5850	5850	5838	5866	5866	5854	5882	5882	5870	5902	5902	5890
5930	5930	5838	5946	5957	5854	5962	5962	5870	5971	5974	5890
5850	5850	5838	5866	5866	5854	5882	5882	5870	5902	5943	5890
5930	5930	5838	5946	5946	5854	5962	5962	5870	5982	5974	5890
5850	5850	5838	5866	5866	5854	5882	5882	5870	5902	5902	5890
5953	5930	5838	5946	5946	5854	5962	6011	5870	6031	5974	5890
5907	5850	5838	5866	5866	5854	5882	5882	5870	5902	5902	5890
5953	5930	5838	5978	5946	5854	5951	5962	5870	5971	5974	5890
5850	5887	5838	5866	5866	5854	5882	5882	5870	5902	5902	5890
5919	5930	5838	5967	6008	5854	5951	5962	5870	5971	5974	5890
5850	5850	5838	5866	5866	5854	5882	5923	5870	5902	5902	5890
5919	5930	5838	5946	5946	5854	5951	5962	5870	5971	6016	5996

Table A.17: CPU Cycle Data for Experiment 4 Continued.

131,072,000		
Rep 1	Rep 2	Rep 3
5918	5918	5906
5998	5971	5906
5918	5918	5906
5998	5999	5906
5918	5918	5906
5998	5971	5906
5918	5918	5906
5998	5971	5906
5918	5918	5906
6051	5971	5906
5918	5918	5906
6004	5971	5906
5918	5918	5906
5998	5971	5906
5918	5918	5906
5998	5987	5906
5918	5918	5906
5998	5971	5906
5918	5918	5906
5998	5971	5906
5918	5918	5906
5998	5971	5906
5918	5918	5906
5998	5971	5906
5918	5918	5906
5998	5971	5906
5943	5918	5906
5998	5971	5906
5918	5918	5906
6063	6070	5906
5918	5918	5906
5998	5971	5906
5918	5918	5906
5998	5971	5906
5918	5918	5906
5998	5986	5906
5951	5918	5906
5998	5971	5906
5918	5918	5906
5998	5994	5906
5918	5918	5906
5998	5971	5906
5918	5918	5906
5998	5971	5906
5918	5918	5906
5998	5971	5906
5918	5918	5906
5998	6005	5906

## *Appendix B. Pilot Test Data*

This appendix contains the pilot test data used to make decisions in designing the TRAPP-2 system. Section B.1.1 investigates the BRAM versus SDRAM memory scheme processing times. Section B.1.2 determines the additional CPU cycles required to detect DNS packets. Section B.1.3 compares the number of CPU cycles required to sdbm hash the smallest and largest SIP/DNS domains. Section B.1.4 compares the number of CPU cycles required to copy the smallest and largest packet sizes into a software buffer.

### ***B.1 Results of Pilot Studies***

*B.1.1 BRAM versus SDRAM Memory Scheme.* Table B.1 contains the number of CPU cycles required to process a SIP packet for the BRAM and SDRAM memory configurations. For each configuration, 50 packets are sent to the apparatus and the number of CPU cycles required to process the packet are recorded in the table. The SDRAM memory scheme averages 16,076 CPU cycles, and the BRAM memory scheme averages 15,299 CPU cycles. The average difference between the memory schemes is 777 CPU cycles. The design decision is to accept the average increase of 777 CPU cycles for the larger addressable memory range, thus the SDRAM memory scheme is selected.

Table B.1: CPU Cycles Used to Process a SIP Packet.

Packet	BRAM	SDRAM
1	19554	21119
2	15304	16037
3	15227	15949
4	15367	15979
5	15272	15910
6	15296	16007
7	15259	15959
8	15354	15989
9	15320	15978
10	15324	15921
11	15147	16012
12	15321	16019
13	15337	15911
14	15274	15915
15	15196	15977
16	15397	15958
17	15275	15933
18	15242	15973
19	15256	16025
20	15361	15992
21	15117	16019
22	15158	15911
23	14876	16009
24	14989	15938
25	15289	15979
26	15406	16065
27	15409	15911
28	15402	15943
29	15179	15910
30	14770	15917
31	14789	16005
32	14854	16005
33	15441	15938
34	15334	15984
35	15705	16013
36	15947	15955
37	15120	15989
38	15056	15988
39	15057	16019
40	15111	15929
41	15120	15921
42	15044	15992
43	15057	15961
44	15111	15981
45	15120	16021
46	15056	15968
47	15057	16019
48	15111	15959
49	15121	15985
50	15041	16025
Average	15,299	16,076



*B.1.2 DNS Packet Detection.* Table B.2 contains the number of cycles required to identify a DNS packet. 50 packets are sent to the TRAPP-2 system, and the number of CPU cycles required to identify a DNS packet is recorded in the table. The system averages 23 CPU cycles to identify a DNS packet.

Table B.2: CPU Cycles Used to Identify a DNS Packet.

Packet	CPU Cycles
1	23
2	23
3	23
4	23
5	23
6	23
7	23
8	23
9	23
10	23
11	23
12	23
13	23
14	23
15	23
16	23
17	23
18	23
19	23
20	23
21	23
22	23
23	23
24	23
25	23
26	23
27	23
28	23
29	23
30	23
31	23
32	23
33	23
34	23
35	23
36	23
37	23
38	23
39	23
40	23
41	23
42	23
43	23
44	23
45	23
46	23
47	23
48	23
49	23
50	23
Average	23

*B.1.3 sdbm Hashing Times.* Table B.3 contains the number of CPU cycles required to process the smallest and largest SIP domain address. 50 packets with a six-character domain and 50 packets with a 212-character domain are sent to the TRAPP-2 system; the number of CPU cycles required to process the packets are recorded in the table. The system averages 86 CPU cycles to sdbm hash a six-character domain and 1195 to sdbm hash a 212-character domain.

Table B.3: CPU Cycles Used to sdbm hash a SIP Packet.

Packet	Small Domain	Large Domain
1	86	1195
2	86	1195
3	86	1195
4	86	1195
5	86	1195
6	86	1195
7	86	1195
8	86	1195
9	86	1195
10	86	1195
11	86	1195
12	86	1195
13	86	1195
14	86	1195
15	86	1195
16	86	1195
17	86	1195
18	86	1195
19	86	1195
20	86	1195
21	86	1195
22	86	1195
23	86	1195
24	86	1195
25	86	1195
26	86	1195
27	86	1195
28	86	1195
29	86	1195
30	86	1195
31	86	1195
32	86	1195
33	86	1195
34	86	1195
35	86	1195
36	86	1195
37	86	1195
38	86	1195
39	86	1195
40	86	1195
41	86	1195
42	86	1195
43	86	1195
44	86	1195
45	86	1195
46	86	1195
47	86	1195
48	86	1195
49	86	1195
50	86	1195
Average	86	1195

*B.1.4 Packet Size Transfer Times.* Table B.4 below contains the number of CPU cycles required to transfer a packet from the TRAPP-2 system into the software buffer. 50 packets with a size of 67 bytes and 50 packets with a size of 1,500 bytes are sent to the TRAPP-2 system; the number of CPU cycles required to process the packets are recorded in the table. The system averages 999 CPU cycles to transfer a 67 byte packet and 18,112 CPU cycles to transfer a 1,500 byte packet.

Table B.4: CPU Cycles Used to Copy Smallest versus Largest Packet.

Packet	67-Byte Packet	1500-Byte Packet
1	999	18112
2	999	18112
3	999	18112
4	999	18112
5	999	18112
6	999	18112
7	999	18112
8	999	18112
9	999	18112
10	999	18112
11	999	18112
12	999	18112
13	999	18112
14	999	18112
15	999	18112
16	999	18112
17	999	18112
18	999	18112
19	999	18112
20	999	18112
21	999	18112
22	999	18112
23	999	18112
24	999	18112
25	999	18112
26	999	18112
27	999	18112
28	999	18112
29	999	18112
30	999	18112
31	999	18112
32	999	18112
33	999	18112
34	999	18112
35	999	18112
36	999	18112
37	999	18112
38	999	18112
39	999	18112
40	999	18112
41	999	18112
42	999	18112
43	999	18112
44	999	18112
45	999	18112
46	999	18112
47	999	18112
48	999	18112
49	999	18112
50	999	18112
Average	999	18112

## *Appendix C. Constructing the TRAPP-2 System Hardware*

This appendix contains the step-by-step guide to constructing the hardware portion of the TRAPP-2 system. Section C.1 provides a description of the hardware used in the TRAPP-2 system. Section C.2 covers the steps used to construct the TRAPP-2 system using the Base System Builder in Xilinx Platform Studio, version 11.4. Section C.3 details the required software modifications in the hardware files to convert the Ethernet controller from 100 Mbps (Media Independent Interface) to 1000 Mbps (Reduced Gigabit Media Independent Interface v2.0).

### ***C.1 Hardware Description***

*C.1.1 Microprocessor.* The on-chip PowerPC 440 processor is used in the TRAPP-2 system. The processor executes the software application.

*C.1.2 Synchronous Dynamic Random Access Memory.* Two 512 MB SDRAM modules are used for the TRAPP-2 system. The first module is formatted as Xilinx Memory File System to temporarily store the hash file before the hashes are transferred to the second SDRAM module. The second module does not have a file system and contains the actual hashes for the hash list. The second module also stores the log file during sniffing. At the completion of sniffing, the log file is transferred to the first SDRAM module so it can be downloaded from the FPGA board.

*C.1.3 Block Random Access Memory.* For this implementation, one 128-kilobyte BRAM is used. The BRAM block contains the bootup software code, data and instruction memory, as well as the stack and heap.

*C.1.4 XPS Hard Ethernet Media Access Controller.* This is the board's Ethernet connection. The Ethernet controller is set to promiscuous mode to receive all packets traversing the network. The Ethernet controller is configured as a RGMII v2.0, capable of operating at 1000 Mbps. More details on programming the Ethernet controller to operate as a RGMII interface can be found in Section C.3.

*C.1.5 RS232 Universal Asynchronous Receiver/Transmitter.* The RS232 interface serves two purposes. The first is to output general information about board initialization and operational status as well as detect user input to stop sniffing. The second purpose is to upload hash files to the board and download the Wireshark-compatible log file. This is accomplished with the xmodem protocol through the TeraTerm Virtual Terminal program [Ter09].

*C.1.6 XPS Timer.* The timer is used to take timestamps for calculating the packet processing time, measured in CPU cycles. Only one of the two available timers is used.

## ***C.2 Component Configuration***

This section provides a step-by-step guide to construct the TRAPP-2 system using the Base System Builder in Xilinx Platform Studio, version 11.4.

1. To begin Open Xilinx Platform Studio, Click on **File**, then **New Project**. A window will appear like the one in Figure C.1.

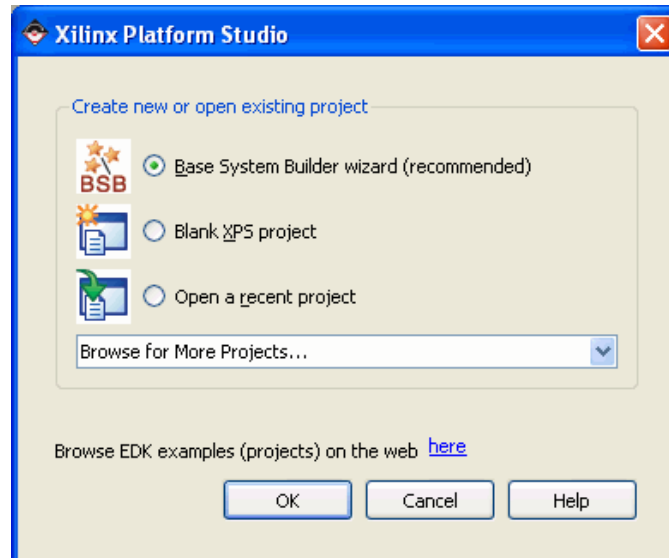


Figure C.1: The Project Creation Options Window.

2. Name the project file, as seen in Figure C.2. Click “OK” to continue.

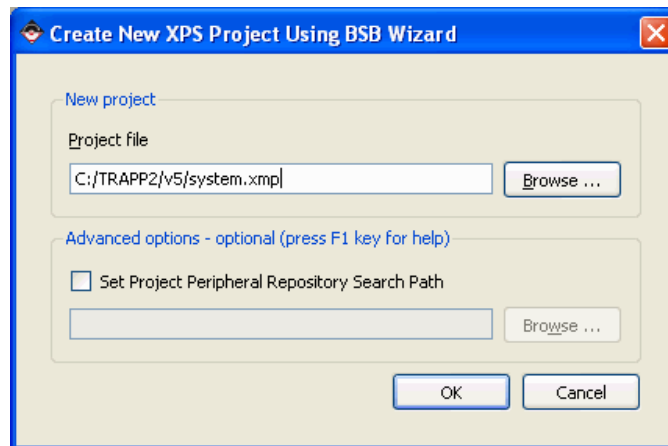


Figure C.2: The Project Creation and Repository Selection Window.



3. Select the “I would like to create a new design” radio button as seen in Figure C.3. Click “Next” to continue.

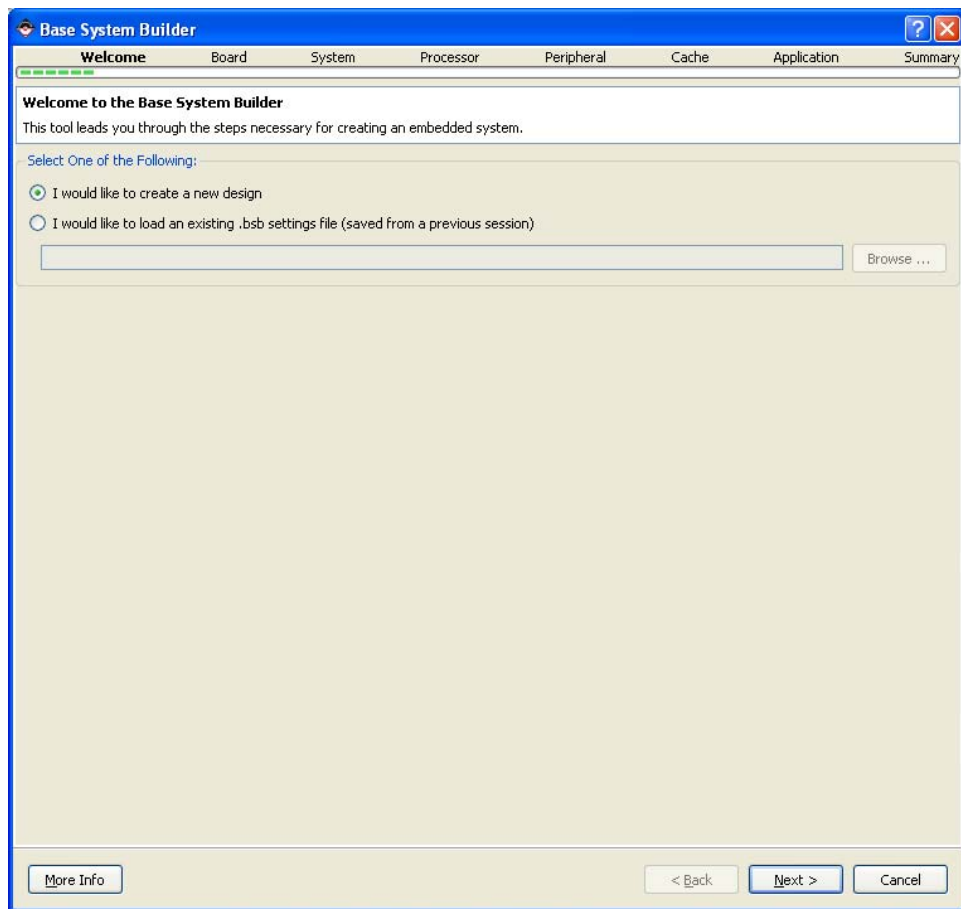


Figure C.3: The Base System Builder Design Selection Window.

4. Select the “I would like to create a system for the following development board” radio button. Choose “Xilinx” for the Board Vendor, “Virtex 5 ML510 Evaluation Platform” for the Board Name, and “C” for the Board Revision, as seen in Figure C.4. Click “Next” to continue.

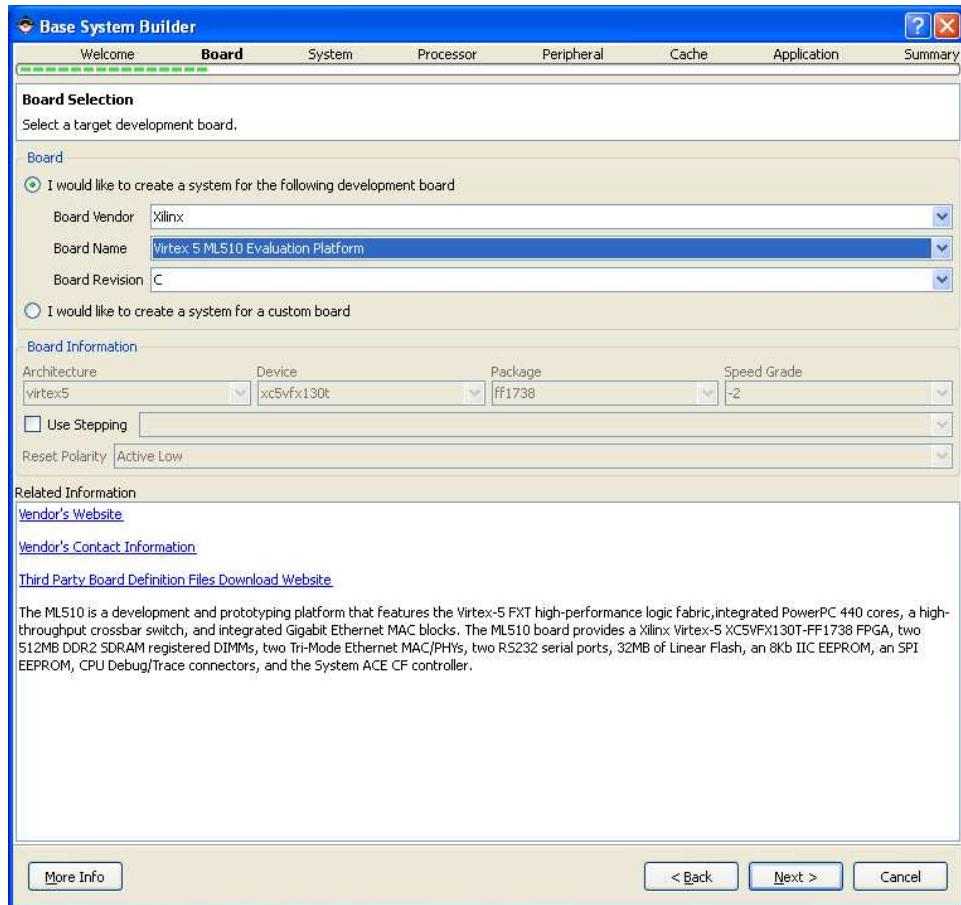


Figure C.4: The Board Selection Window.

5. Select the “Single-Processor System” radio button as seen in Figure C.5. Click “Next” to continue.

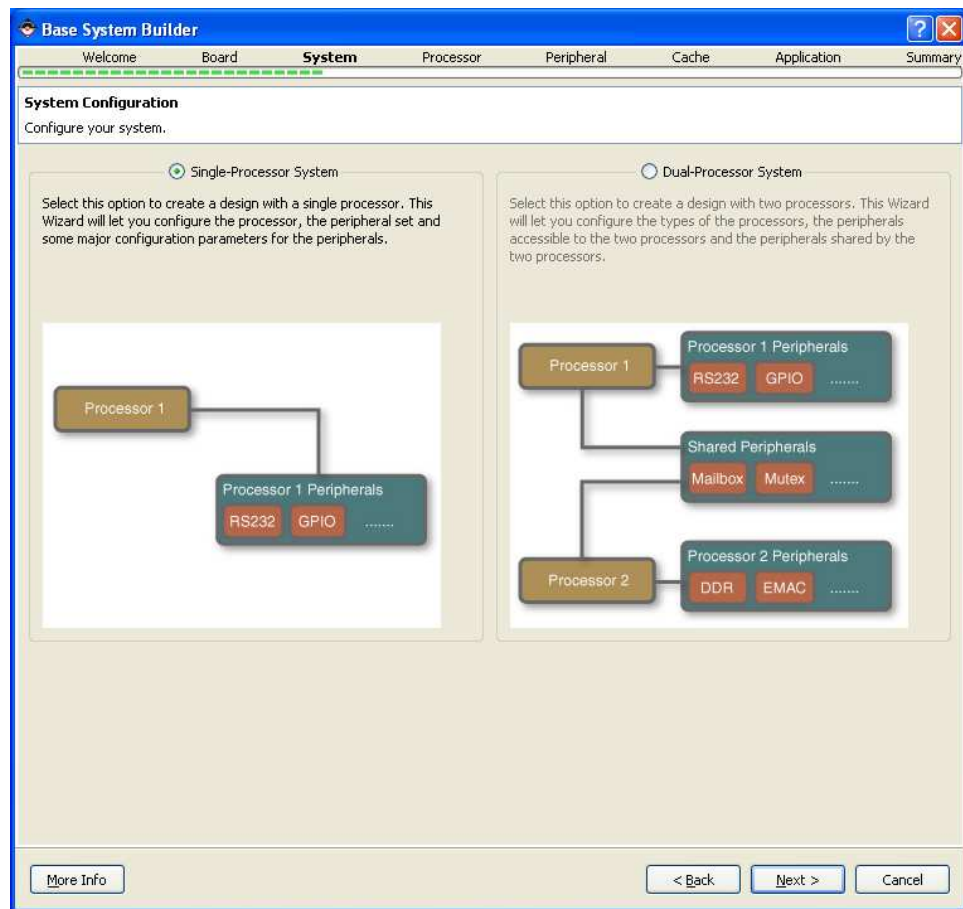


Figure C.5: The Processor Selection Window.

6. Select the “PowerPC” for the Processor Type, “400.00” for Processor Clock Frequency, and “100.00” for the Bus Clock Frequency, as seen in Figure C.6. Click “Next” to continue.

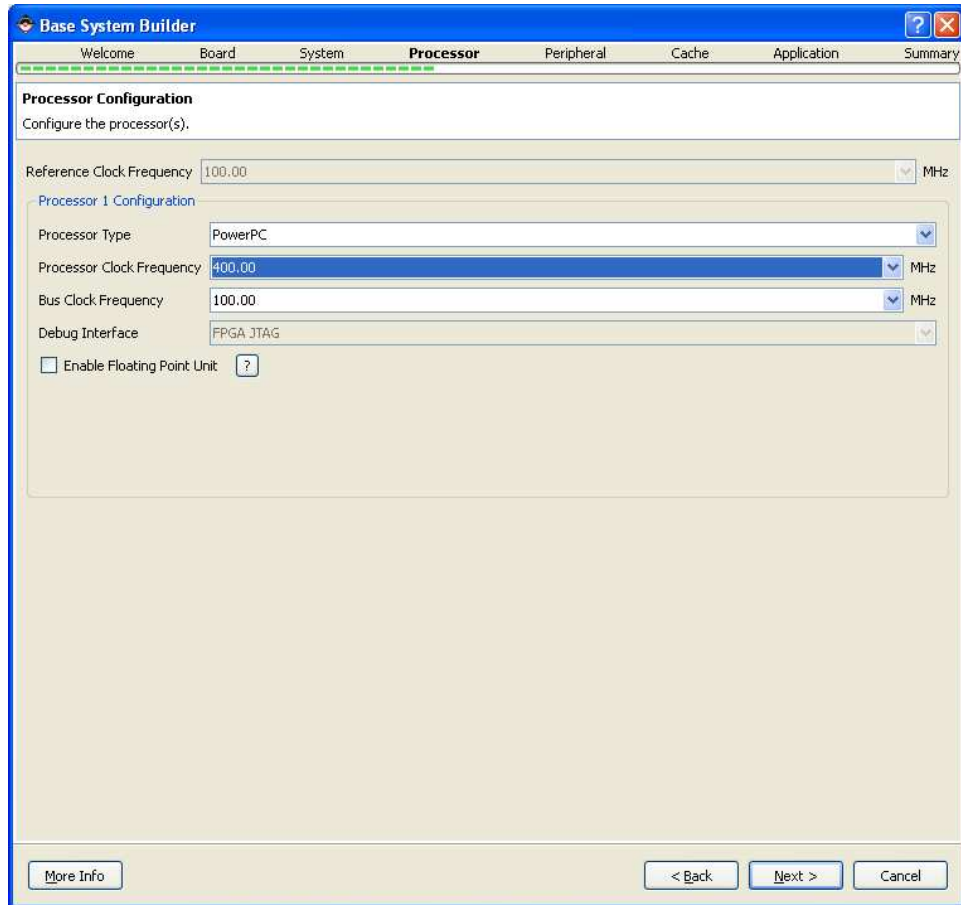


Figure C.6: The Processor Configuration Window.

7. Add the following peripherals, ensuring to change certain options (shown in parenthesis) from the dropdown menus, as seen in Figure C.7. The peripherals include:

- DDR2\_SDRAM\_DIMM0
- DDR2\_SDRAM\_DIMM1
- Hard\_Ethernet\_MAC
- RS232\_Uart\_1 (Set Baud Rate to 115200)
- xps\_bram\_if\_cntlr\_0 (Set Size to 128 KB)
- xps\_timer\_0

Click “Next” to continue.

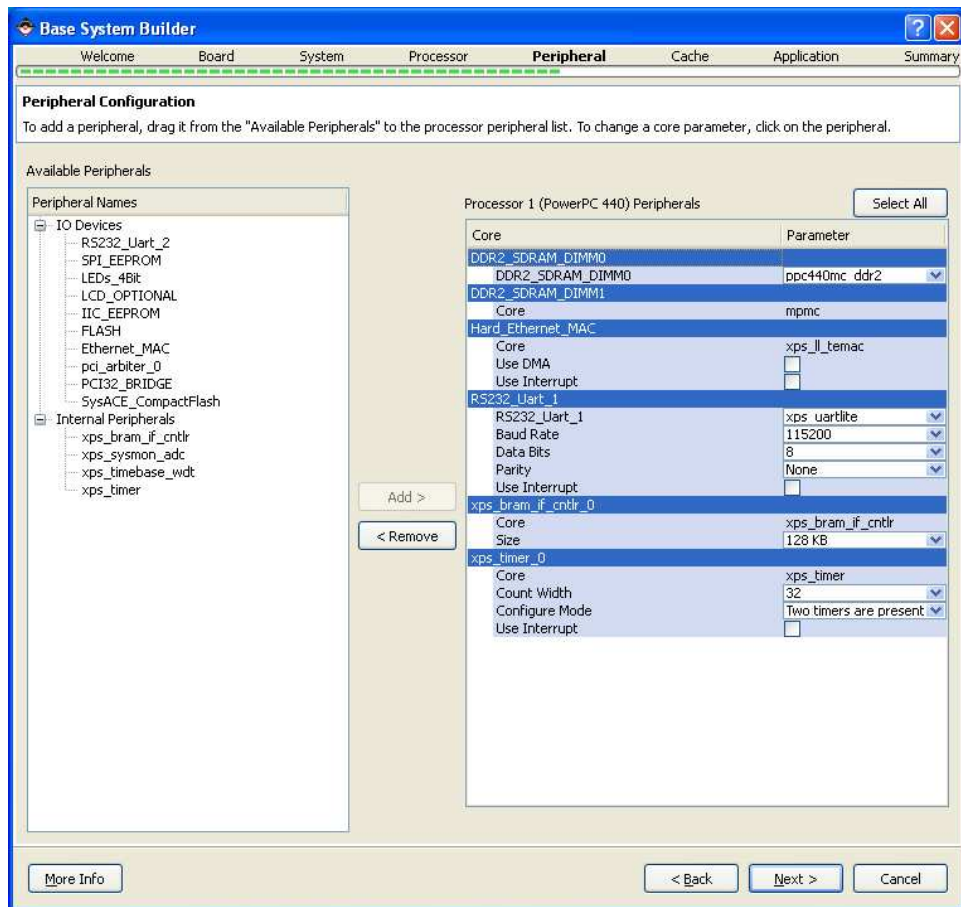


Figure C.7: The Peripheral Configuration Window.

8. Check every box for enabling the Data and Instruction caches for the processor, as seen in Figure C.8. Click “Next” to continue.

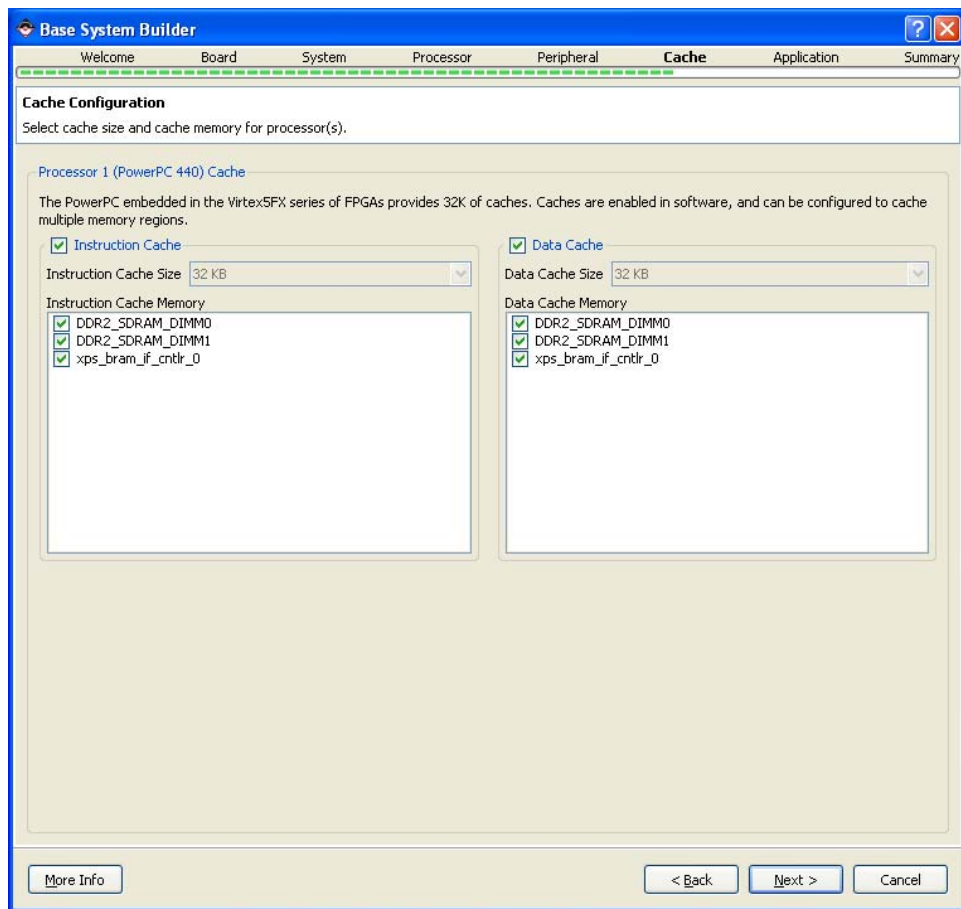


Figure C.8: The Processor Cache Configuration Window.

9. The Memory and Peripheral Test Applications are optional, but highly recommended to ensure the board is functioning properly. Select `xps.bram.if.cntrl_0` as the location to store the test applications, as seen in Figure C.9. Click “Next” to continue.

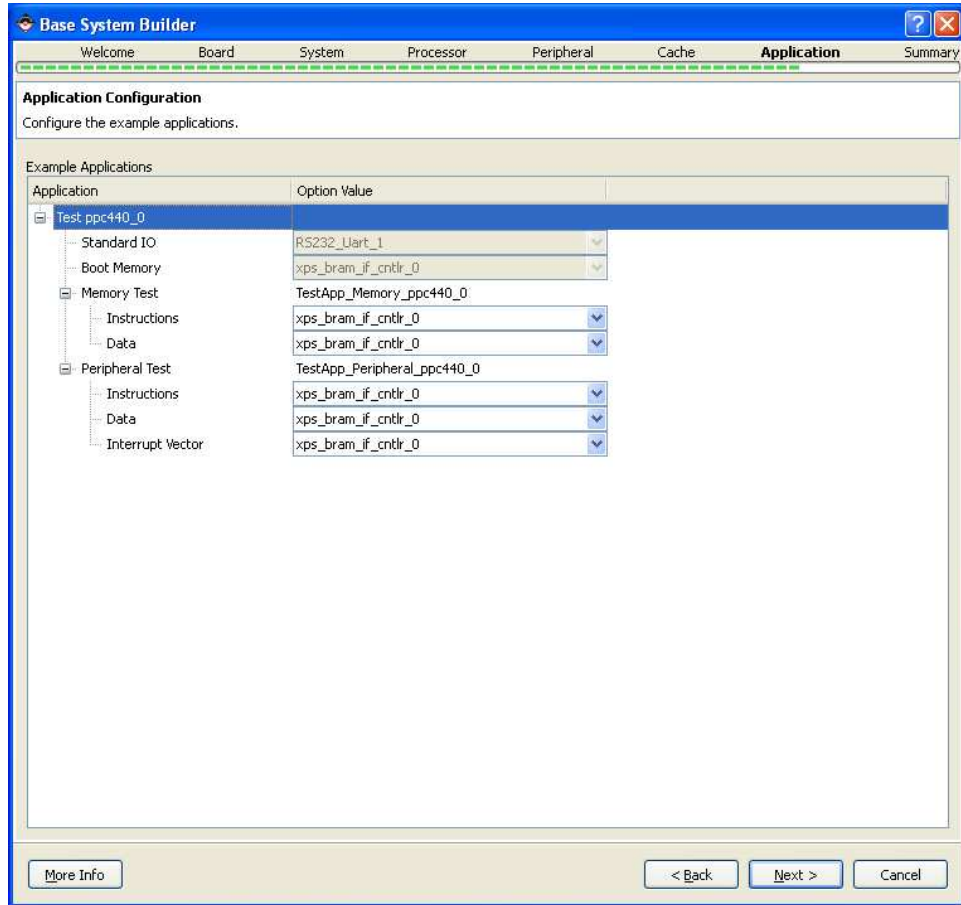


Figure C.9: The Application Selection Window.

10. This window summarizes the system being built, as seen in Figure C.10. Click “Finish” to continue.

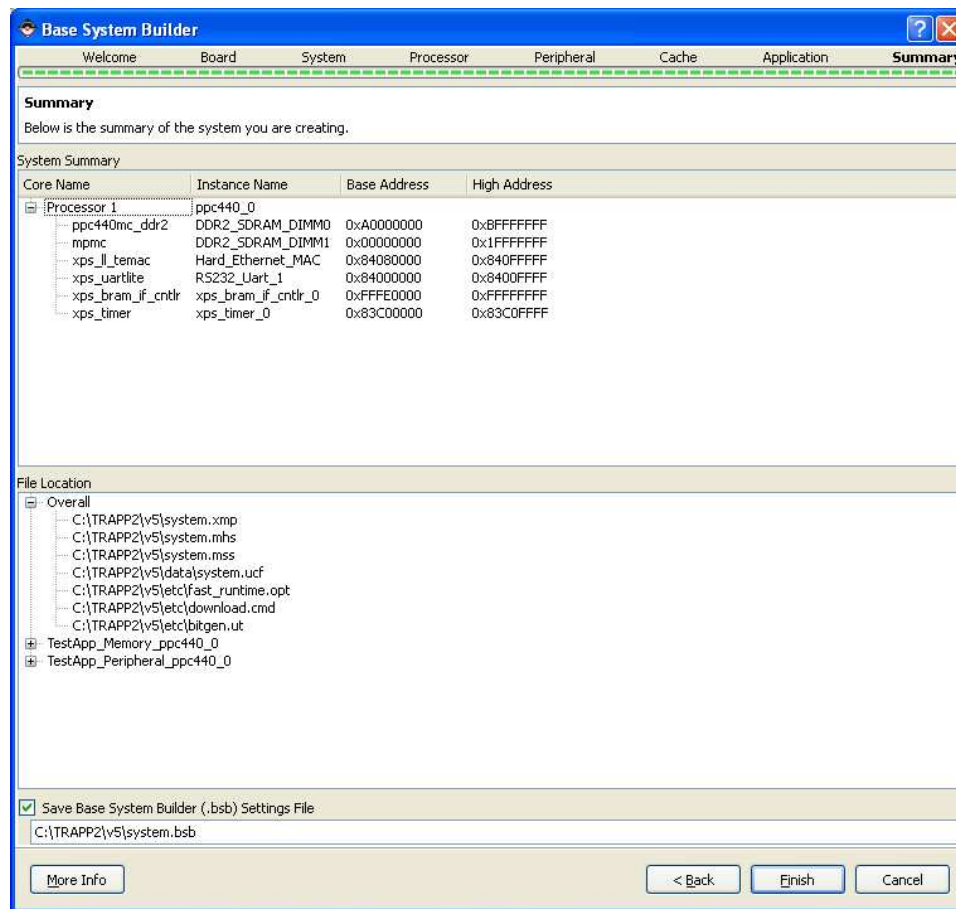


Figure C.10: The Summary Configuration Window.



11. After completing the Base System Builder, a window pops up asking about additional configuration settings. Select the “Configure drivers and libraries (Software Platform)” radio button, as seen in Figure C.11. Click “OK” to continue.

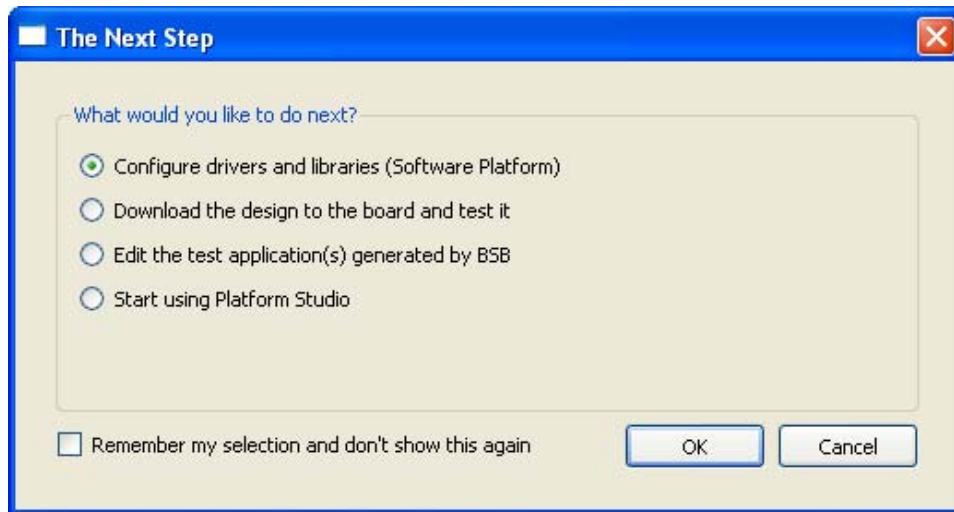


Figure C.11: The Configure Libraries and Drivers Window.

12. Highlight “Software Platform” in the left side column. Place a check in the “xilmfs” checkbox, as seen in Figure C.12. Click “OK” to continue.

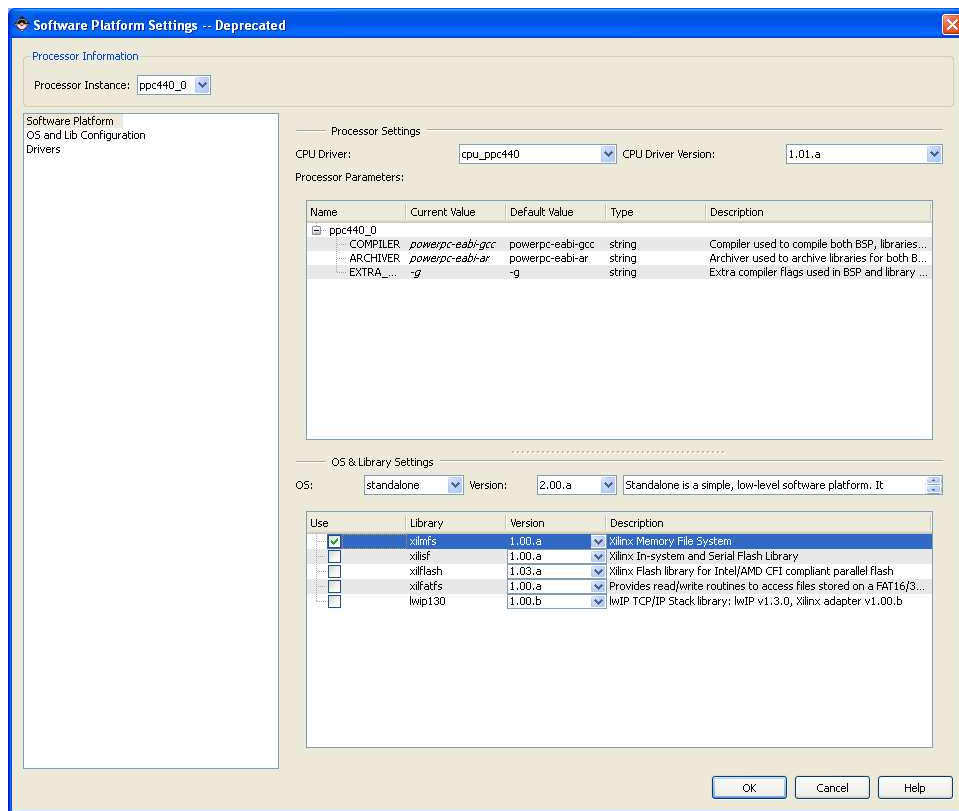


Figure C.12: The Software Platform Settings Window.

13. Highlight “OS and Lib Configuration” in the left side column. Change the following options: numbytes (100,000,000), base\_address = 0x0 (this is the base address of the SDRAM that holds the Xilinx Memory File System), init\_type (MFSINIT\_NEW), need\_utils (true), as seen in Figure C.13. Click “OK” to continue. Ensure these values are accurately reflected in the .mss file.

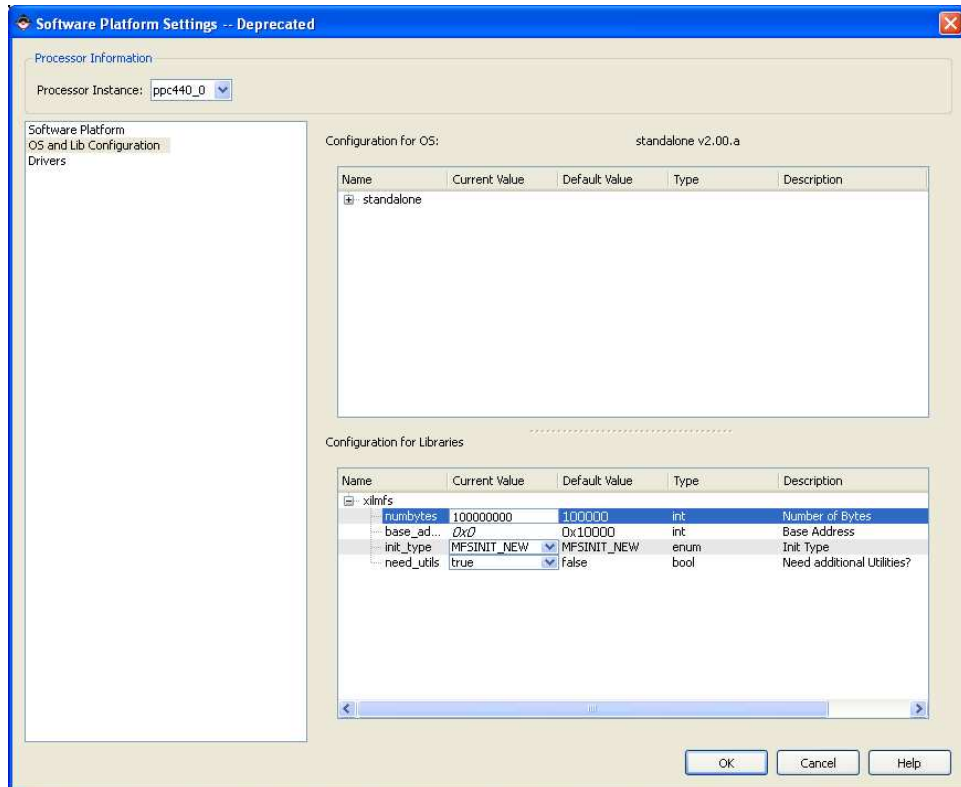


Figure C.13: The Software Platform Settings OS and Lib Configuration Window.

### ***C.3 Converting from MII to RGMII v2.0***

This section details the software modifications in the hardware files to convert the Ethernet controller from 100 Mbps (MII) to 1000 Mbps (RGMII v2.0).

1. Configure HARD\_ETHERNET\_MAC IP

```
C_NUM_IDELAYCTRL=1
C_IDELAYCTRL_LOC=NOT_SET
Physical Interface Type = RGMII V2.0
RX FIFO Depth of TEMAC0 = 32768B
```

2. In the .mhs file, make the following modifications.

Remove all MII\* from external ports and add the following lines:

```
PORT Hard_Ethernet_MAC_RGMII_TXD_0_pin = Hard_Ethernet_MAC_RGMII_TXD_0, DIR = 0, VEC = [3:0]
PORT Hard_Ethernet_MAC_RGMII_TXC_0_pin = Hard_Ethernet_MAC_RGMII_TXC_0, DIR = 0
PORT Hard_Ethernet_MAC_RGMII_TX_CTL_0_pin = Hard_Ethernet_MAC_RGMII_TX_CTL_0, DIR = 0
PORT Hard_Ethernet_MAC_RGMII_RXD_0_pin = Hard_Ethernet_MAC_RGMII_RXD_0, DIR = I, VEC = [3:0]
PORT Hard_Ethernet_MAC_RGMII_RX_CTL_0_pin = Hard_Ethernet_MAC_RGMII_RX_CTL_0, DIR = I
PORT Hard_Ethernet_MAC_RGMII_RXC_0_pin = Hard_Ethernet_MAC_RGMII_RXC_0, DIR = I
```

In the xps\_ll\_temac section of the .mhs file, PORT REFCLK must be connected to a 200MHz clock. The name may be different between designs.

Add:

```
PORT GTX_CLK_0 = clk_125mhz
PORT REFCLK = clk_200_0000MHzPLL0
PORT RGMII_RXD_0 = Hard_Ethernet_MAC_RGMII_RXD_0
PORT RGMII_RX_CTL_0 = Hard_Ethernet_MAC_RGMII_RX_CTL_0
PORT RGMII_RXC_0 = Hard_Ethernet_MAC_RGMII_RXC_0
PORT RGMII_TXC_0 = Hard_Ethernet_MAC_RGMII_TXC_0
PORT RGMII_TX_CTL_0 = Hard_Ethernet_MAC_RGMII_TX_CTL_0
PORT RGMII_TXD_0 = Hard_Ethernet_MAC_RGMII_TXD_0
```

In the clock\_generator section, a 125MHz clock must be added. The number of the clock depends on the number of other clocks in your design. Since four clocks already existed, the new one is C\_CLKOUT5.

Add:

```
PARAMETER C_CLKOUT5_FREQ = 125000000
PARAMETER C_CLKOUT5_PHASE = 0
PARAMETER C_CLKOUT5_GROUP = NONE
PARAMETER C_CLKOUT5_BUF = TRUE
PORT CLKOUT5 = clk_125mhz
```

3. In the .ucf file, make the following modifications.

Remove:

```
Net fpga_0_Hard_Ethernet_MAC_MII_TXD_0_pin<3> LOC=AN31 | IOSTANDARD = LVCMOS25 | SLEW = FAST
| DRIVE = 6;
Net fpga_0_Hard_Ethernet_MAC_MII_TXD_0_pin<2> LOC=AR32 | IOSTANDARD = LVCMOS25 | SLEW = FAST
| DRIVE = 6;
Net fpga_0_Hard_Ethernet_MAC_MII_TXD_0_pin<1> LOC=AP32 | IOSTANDARD = LVCMOS25 | SLEW = FAST
| DRIVE = 6;
Net fpga_0_Hard_Ethernet_MAC_MII_TXD_0_pin<0> LOC=AR33 | IOSTANDARD = LVCMOS25 | SLEW = FAST
| DRIVE = 6;
Net fpga_0_Hard_Ethernet_MAC_MII_TX_EN_0_pin LOC=AP31 | IOSTANDARD = LVCMOS25 | SLEW = FAST
| DRIVE = 6;
Net fpga_0_Hard_Ethernet_MAC_MII_TX_ER_0_pin LOC=AT31 | IOSTANDARD = LVCMOS25 | SLEW = FAST
| DRIVE = 6;
Net fpga_0_Hard_Ethernet_MAC_MII_RXD_0_pin<3> LOC=AM33 | IOSTANDARD = LVCMOS25;
Net fpga_0_Hard_Ethernet_MAC_MII_RXD_0_pin<2> LOC=AK33 | IOSTANDARD = LVCMOS25;
Net fpga_0_Hard_Ethernet_MAC_MII_RXD_0_pin<1> LOC=AJ33 | IOSTANDARD = LVCMOS25;
Net fpga_0_Hard_Ethernet_MAC_MII_RXD_0_pin<0> LOC=AJ32 | IOSTANDARD = LVCMOS25;
Net fpga_0_Hard_Ethernet_MAC_MII_RX_DV_0_pin LOC=AN33 | IOSTANDARD = LVCMOS25;
Net fpga_0_Hard_Ethernet_MAC_MII_RX_ER_0_pin LOC=AP33 | IOSTANDARD = LVCMOS25;
Net fpga_0_Hard_Ethernet_MAC_MII_RX_CLK_0_pin LOC=J17 | IOSTANDARD = LVCMOS25;
Net fpga_0_Hard_Ethernet_MAC_MII_TX_CLK_0_pin LOC=M26 | IOSTANDARD = LVCMOS25;
```

Add:

```
Net Hard_Ethernet_MAC_RGMII_TXD_0_pin<3> LOC = AN31 | IOSTANDARD=LVCMOS25 | SLEW=FAST
| DRIVE = 24;
Net Hard_Ethernet_MAC_RGMII_TXD_0_pin<2> LOC = AR32 | IOSTANDARD=LVCMOS25 | SLEW=FAST
| DRIVE = 24;
Net Hard_Ethernet_MAC_RGMII_TXD_0_pin<1> LOC = AP32 | IOSTANDARD=LVCMOS25 | SLEW=FAST
| DRIVE = 24;
Net Hard_Ethernet_MAC_RGMII_TXD_0_pin<0> LOC = AR33 | IOSTANDARD=LVCMOS25 | SLEW=FAST
| DRIVE = 24;
Net Hard_Ethernet_MAC_RGMII_TX_CTL_0_pin LOC = AP31 | IOSTANDARD=LVCMOS25 | SLEW=FAST
```

```

| DRIVE = 24;
Net Hard_Ethernet_MAC_RGMII_TXC_0_pin LOC = AM31 | IOSTANDARD =LVCMOS25 | SLEW = FAST
| DRIVE = 6;
Net Hard_Ethernet_MAC_RGMII_RXD_0_pin<3> LOC = AM33 | IOSTANDARD=LVCMOS25;
Net Hard_Ethernet_MAC_RGMII_RXD_0_pin<2> LOC = AK33 | IOSTANDARD=LVCMOS25;
Net Hard_Ethernet_MAC_RGMII_RXD_0_pin<1> LOC = AJ33 | IOSTANDARD=LVCMOS25;
Net Hard_Ethernet_MAC_RGMII_RXD_0_pin<0> LOC = AJ32 | IOSTANDARD=LVCMOS25;
Net Hard_Ethernet_MAC_RGMII_RX_CTL_0_pin LOC = AN33 | IOSTANDARD=LVCMOS25;
Net Hard_Ethernet_MAC_RGMII_RXC_0_pin LOC=J17 | IOSTANDARD=LVCMOS25;

```

Remove:

```

##### Hard_Ethernet_MAC
NET "*Hard_Ethernet_MAC/LlinkTemac0_CLK*" TNM_NET = "LLCLK0";
#name of signal connected to TEMAC LlinkTemac0_CLK input
NET "*Hard_Ethernet_MAC/SPLB_Clk*" TNM_NET = "PLBCLK";
#name of signal connected to TEMAC SPLB_Clk input

# EMACO TX Client Clock
NET "*Hard_Ethernet_MAC/TxClientClk_0" TNM_NET = "clk_client_tx0";
TIMEGRP "mii_client_clk_tx0" = "clk_client_tx0";
TIMESPEC "TS_mii_client_clk_tx0" = PERIOD "mii_client_clk_tx0"
7500 ps HIGH 50 %;

# EMACO RX Client Clock
NET "*Hard_Ethernet_MAC/RxClientClk_0" TNM_NET = "clk_client_rx0";
TIMEGRP "mii_client_clk_rx0" = "clk_client_rx0";
TIMESPEC "TS_gmii_client_clk_rx0" = PERIOD "gmii_client_clk_rx0" 7500 ps HIGH 50 %;

# EMACO RX PHY Clock
NET "*Hard_Ethernet_MAC/MII_RX_CLK_0*" TNM_NET = "phy_clk_rx0";
TIMEGRP "mii_clk_phy_rx0" = "phy_clk_rx0";
TIMESPEC "TS_mii_clk_phy_rx0" = PERIOD "mii_clk_phy_rx0" 40000 ps HIGH 50 %;

# EMACO TX MII 10/100 PHY Clock
NET "*Hard_Ethernet_MAC/MII_TX_CLK_0*" TNM_NET = "clk_mii_tx_clk0";
TIMESPEC "TS_mii_tx_clk0" = PERIOD "clk_mii_tx_clk0" 40000 ps HIGH 50 %;

# MII Receiver Constraints: place flip-flops in IOB
INST "*mii0*RXD_TO_MAC*" IOB = TRUE;
INST "*mii0*RX_DV_TO_MAC*" IOB = TRUE;
INST "*mii0*RX_ER_TO_MAC*" IOB = TRUE;

# PHY spec: 10ns setup time, 10ns hold time
# Assumes equal length board traces

NET "fpga_0_Hard_Ethernet_MAC_MII_RXD_0_pin(?)" TNM = "mii_rx_0";
NET "fpga_0_Hard_Ethernet_MAC_MII_RX_DV_0_pin" TNM = "mii_rx_0";
NET "fpga_0_Hard_Ethernet_MAC_MII_RX_ER_0_pin" TNM = "mii_rx_0";

TIMEGRP "mii_rx_0" OFFSET = IN 10 ns VALID 20 ns BEFORE
"fpga_0_Hard_Ethernet_MAC_MII_RX_CLK_0_pin";

# MII Transmitter Constraints: place flip-flops in IOB
INST "*mii0*MII_TXD_?" IOB = TRUE;
INST "*mii0*MII_TX_EN" IOB = TRUE;
INST "*mii0*MII_TX_ER" IOB = TRUE;

TIMESPEC TS_PLB_2_TXPHY0 = FROM PLBCLK TO clk_phy_tx0 40000 ps DATAPATHONLY; #constant
value based on Ethernet clock
TIMESPEC TS_RXPHY0_2_PLB = FROM phy_clk_rx0 TO PLBCLK 10000 ps DATAPATHONLY; #varies

```

```

based on period of PLB clock

TIMESPEC "TS_LL_CLK0_2_RX_CLIENT_CLK0" = FROM LLCLK0 TO clk_client_rx0 8000 ps
DATAPATHONLY; #constant value based on Ethernet clock
TIMESPEC "TS_LL_CLK0_2_TX_CLIENT_CLK0" = FROM LLCLK0 TO clk_client_tx0 8000 ps
DATAPATHONLY; #constant value based on Ethernet clock
TIMESPEC "TS_RX_CLIENT_CLK0_2_LL_CLK0" = FROM clk_client_rx0 TO LLCLK0 10000 ps
DATAPATHONLY; #varies based on period of LocalLink clock
TIMESPEC "TS_TX_CLIENT_CLK0_2_LL_CLK0" = FROM clk_client_tx0 TO LLCLK0 10000 ps
DATAPATHONLY; #varies based on period of LocalLink clock

net "*/hrst*" TIG;

```

Add:

```

##### Hard_Ethernet_MAC

# EMACO TX Client Clock
NET "*/RGMII_TX_CTL_0*" TNM_NET = "clk_client_tx0";
TIMEGRP "rgmii_client_clk_tx0" = "clk_client_tx0";
TIMESPEC "TS_rgmii_client_clk_tx0" = PERIOD "rgmii_client_clk_tx0" 7800 ps HIGH 50 %;

# EMACO RX Client Clock
NET "*/RGMII_RX_CTL_0*" TNM_NET = "clk_client_rx0";
TIMEGRP "rgmii_client_clk_rx0" = "clk_client_rx0";
TIMESPEC "TS_rgmii_client_clk_rx0" = PERIOD "rgmii_client_clk_rx0" 7800 ps HIGH 50 %;

# EMACO TX PHY Clock
NET "*/RGMII_TXC_0*" TNM_NET = "clk_phy_tx0";
TIMEGRP "rgmii_phy_clk_tx0" = "clk_phy_tx0";
TIMESPEC "TS_rgmii_phy_clk_tx0" = PERIOD "rgmii_phy_clk_tx0" 7800 ps HIGH 50 %;

# EMACO RX PHY Clock
NET "*/RGMII_RXC_0*" TNM_NET = "clk_phy_rx0";
TIMEGRP "rgmii_clk_phy_rx0" = "clk_phy_rx0";
TIMESPEC "TS_rgmii_clk_phy_rx0" = PERIOD "rgmii_clk_phy_rx0" 7800 ps HIGH 50 %;

# Set the IDELAY values on the data inputs.
# Please modify to suit your design.
INST "*/rgmii0?rgmii_rx_ctl_delay" IOBDELAY_TYPE = FIXED;
INST "*/rgmii0?rgmii_rx_d0_delay" IOBDELAY_TYPE = FIXED;
INST "*/rgmii0?rgmii_rx_d1_delay" IOBDELAY_TYPE = FIXED;
INST "*/rgmii0?rgmii_rx_d2_delay" IOBDELAY_TYPE = FIXED;
INST "*/rgmii0?rgmii_rx_d3_delay" IOBDELAY_TYPE = FIXED;
INST "*/rgmii_rxc0_delay" IOBDELAY_TYPE = FIXED;
INST "*/rgmii0?rgmii_rx_ctl_delay" IDELAY_VALUE = 25;
INST "*/rgmii0?rgmii_rx_d0_delay" IDELAY_VALUE = 25;
INST "*/rgmii0?rgmii_rx_d1_delay" IDELAY_VALUE = 25;
INST "*/rgmii0?rgmii_rx_d2_delay" IDELAY_VALUE = 25;
INST "*/rgmii0?rgmii_rx_d3_delay" IDELAY_VALUE = 25;
INST "*/rgmii_rxc0_delay" IDELAY_VALUE = 0;
NET "*/LlinkTemac0_CLK*" TNM_NET = "LLCLK";
TIMESPEC "TS_LL_CLK0_2_RX_CLIENT_CLK0" = FROM LLCLK0 TO clk_client_rx0 8000 ps DATAPATHONLY;
TIMESPEC "TS_LL_CLK0_2_TX_CLIENT_CLK0" = FROM LLCLK0 TO clk_client_tx0 8000 ps DATAPATHONLY;
TIMESPEC "TS_RX_CLIENT_CLK0_2_LL_CLK0" = FROM clk_client_rx0 TO LLCLK0 8000 ps DATAPATHONLY;
TIMESPEC "TS_TX_CLIENT_CLK0_2_LL_CLK0" = FROM clk_client_tx0 TO LLCLK0 8000 ps DATAPATHONLY;

```

## Bibliography

- AL01. Paul Albitz and Cricket Liu. *DNS and BIND*. O'Reilly, fourth edition, 2001.
- Bay09. Pirate Bay. Home Page, May 2009. <http://thepiratebay.org>.
- BEPW02. P. Biddle, P. England, M. Peinado, and B Willman. The Darknet and the Future of Content Distribution. *Proceedings of the 2002 ACM Workshop on Digital Rights Management*, 2002.
- CCM<sup>+</sup>07. K. P. Chow, K. Y. Cheng, L. Y. Man, Pierre K. Y. Lai, Lucas C. K. Hui, C. F. Chong, K. H. Pun, W. W. Tsang, H. W. Chan, and S. M. Yiu. BTM - An Automated Rule-Based BT Monitoring System for Piracy Detection. *Proceedings of the Second International Conference on Internet Monitoring and Protection*, page 2, 2007.
- Cis02. Cisco. Cisco Introduces New SIP-enabled Voice over IP Solutions, March 2002. [http://newsroom.cisco.com/dlls/prod\\_031102.html](http://newsroom.cisco.com/dlls/prod_031102.html).
- Cis10. Cisco. SIP Successful Call Setup, Apr 2010. [http://www.cisco.com/web/about/ac123/ac147/images/ipj/ipj\\_6-1/session\\_initiation\\_2.gif](http://www.cisco.com/web/about/ac123/ac147/images/ipj/ipj_6-1/session_initiation_2.gif).
- Coh08. Bram Cohen. The BitTorrent Protocol Specification, February 2008. [http://www.bittorrent.org/beps/bep\\_0003.html](http://www.bittorrent.org/beps/bep_0003.html).
- Cou10. CounterPath. X-Lite welcomes you to the world of softphones!, February 2010. <http://www.counterpath.com/x-lite.html>.
- Fel04. Geoff Fellows. Peer-to-peer Networking Issues-An Overview. *Digital Investigation*, pages 3–6, February 2004.
- Fil07. FileShareFreak. Darknets (Private Internet & File Sharing), December 2007. <http://filesharefreak.com/2007/12/16/darknets-private-internet-file-sharing/>.
- Fou10. The Linux Foundation. pktgen, March 2010. <http://www.linuxfoundation.org/collaborate/workgroups/networking/pktgen>.
- FOX09. FOXNews. Report: Marine One Information Found on Computer in Iran, Mar 2009. <http://www.foxnews.com/politics/2009/03/01/report-marine-information-iran/>.
- Fre09. Freenet. What is Freenet?, May 2009. <http://freenetproject.org/whatis.html>.
- Gon05. Yiming Gong. Identifying P2P Users Using Traffic Analysis, July 2005. <http://www.securityfocus.com/infocus/1843>.



- GPW06. Matthew Gebski, Alex Penev, and Raymond Wong. Protocol Identification of Encrypted Network Traffic. *Proceedings of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence*, pages 957–960, December 2006.
- HiP09a. HiPPIE. About HiPPIE, May 2009. <http://hippie.oofle.com/about>.
- HiP09b. HiPPIE. HiPPIE Features, May 2009. <http://hippie.oofle.com/features>.
- ISO09. ISOHunt. Home Page, May 2009. <http://isohunt.com>.
- jhi10. jhind. Welcome to the home of the meanypants projects, April 2010. <http://www.meanypants.com/meanypants/CatchingDNStunnelsWithAI-1.pdf?attredirects=0&d=1>.
- Kah08. Jeremy Kahn. Mumbai Terrorists Relied on New Technology for Attacks, December 2008. [http://www.nytimes.com/2008/12/09/world/asia/09mumbai.html?\\_r=1](http://www.nytimes.com/2008/12/09/world/asia/09mumbai.html?_r=1).
- Kam09. Dan Kaminsky. OzymanDNS 0.1, May 2009. [http://www.doxpara.com/ozymandns\\_src\\_0.1.tgz](http://www.doxpara.com/ozymandns_src_0.1.tgz).
- KBFC04. Thomas Karagiannis, Andre Broido, Michalis Faloutsos, and KC Claffy. Transport Layer Identification of P2P Traffic. *Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement*, pages 121–134, 2004.
- KMHH06. A. Karasaridis, K. Meier-Hellstern, and D. Hoefflin. Detection of DNS Anomalies using Flow Data Analysis. *Global Telecommunications Conference*, pages "1–6", 2006.
- Kry09. Kryo. Iodine by Kryo, July 2009. <http://code.kryo.se/iodine/>.
- Moh09. Tor Mohling. DNS Slides, May 2009. <http://bio3d.colorado.edu/~tor/sadocs/dns/dns.html>.
- MW06. Alok Madhukar and Carey Williamson. A Longitudinal Study of P2P Traffic Classification. *Proceedings of the 14th IEEE International Symposium on Modeling, Analysis, and Simulation*, pages 179–188, September 2006.
- Pea98. Oskar Pearson. DNS Tunnel - Through Bastion Hosts, April 1998. [http://archives.neohapsis.com/archives/bugtraq/1998\\_2/0079.html](http://archives.neohapsis.com/archives/bugtraq/1998_2/0079.html).
- RE10. Remote-Exploit. BackTrack, Feb 2010. <http://www.backtrack-linux.org/downloads/>.
- RFC01. RFC 3174 - US Secure Hash Algorithm 1 (SHA1), September 2001. <http://www.faqs.org/rfcs/rfc3174.html>.
- RFC02. RFC 3261 - SIP: Session Initiation Protocol, June 2002. <http://www.faqs.org/rfcs/rfc3261.html>.

- Riv09. Ronald L. Rivest. Frequently Asked Questions, May 2009.  
<http://people.csail.mit.edu/rivest/faq.html>.
- RKSM08. D.A.L. Romana, S. Kubota, K. Sugitani, and Y. Musashi. DNS Based Spam Bots Detection in a University. *First International Conference on Intelligent Networks and Intelligent Systems*, pages "205–208", 2008.
- RSA09. RSA. What is RC4?, May 2009.  
<http://www.rsa.com/rsalabs/node.asp?id=2250>.
- Sch09. Karl Schrader. An FPGA-Based System for Tracking Digital Information Transmitted Via Peer-to-Peer Protocols. Air Force Institute of Technology, March 2009.
- Sky09. Skype. VoIP Explained, May 2009.  
<http://www.skype.com/help/guides/voip/>.
- Sno10. Snort. Snort FAQ, April 2010. <http://www.snort.org/snort/faq/>.
- SSW04. Subhabrata Sen, Oliver Spatscheck, and Dongmei Wang. Accurate, Scalable In-Network Identification of P2P Traffic Using Application Signatures. *Proceedings of the 13th International Conference on World Wide Web*, pages 512–521, May 2004.
- Sta06. William Stallings. *Cryptography and Network Security: Principles and Practice*. Prentice Hall, fourth edition, 2006.
- SY91. Margo Seltzer and Ozan Yigit. A New Hashing Package for UNIX. *Proceedings of the 1991 Winter Usenix*, pages 1–6, 1991.
- Tcp10. Tcpreplay. Welcome to Tcpreplay, April 2010.  
<http://tcpreplay.synfin.net/>.
- Tec09. VOCAL Technologies. RC4 Encryption Algorithm, May 2009.  
<http://www.vocal.com/cryptography/rc4.html>.
- Ter09. T. Teranishi. Tera Term Home Page, September 2009.  
<http://hp.vector.co.jp/authors/VA002416/teraterm.html>.
- Tor09a. Tor. Tor: Anonymity Online, May 2009. <http://www.torproject.org/>.
- Tor09b. Tor. Tor: Overview, May 2009.  
<http://www.torproject.org/overview.html.en>.
- Tri10. Trixbox. Downloads, Feb 2010. <http://www.trixbox.org/downloads>.
- Tys08. Jeff Tyson. How the Old Napster Worked, June 2008.  
<http://computer.howstuffworks.com/napster2.htm>.
- Ubi08. Ubiquity. Understanding SIP, July 2008.  
[http://www.sipcenter.com/sip.nsf/html/WEBB5YNVK8/\\$File/Ubiquity\\_SIP\\_Overview.pdf](http://www.sipcenter.com/sip.nsf/html/WEBB5YNVK8/$File/Ubiquity_SIP_Overview.pdf).

- uTo10. uTorrent. uTorrent - The Lightweight and Efficient BitTorrent Client, Feb 2010. <http://www.utorrent.com/>.
- Van09. Martin VanHorenbeeck. DNS Tunneling, May 2009. <http://www.daemon.be/maarten/dnstunnel.html>.
- WAS09. WASTE. Overview, May 2009. <http://waste.sourceforge.net/>.
- Wir09. Wireshark. Wireshark Network Protocol Analyzer, May 2009. <http://www.wireshark.org/>.
- Wir10. Wired. Google Hack Attack Was Ultra Sophisticated, New Details Show, January 2010. <http://www.wired.com/threatlevel/2010/01/operation-aurora/>.
- WMM06. Charles Wright, Fabian Monroe, and Gerald Masson. On Inferring Application Protocol Behaviors in Encrypted Network Traffic. *The Journal of Machine Learning Research*, 7:2745–2769, December 2006.
- Xil08. Xilinx. Xilinx University Program Virtex-II Pro Development System, June 2008. <http://www.xilinx.com/products/devkits/XUPV2P.htm>.
- Xil09. Xilinx. Virtex-5 Family Overview, February 2009. [http://www.xilinx.com/support/documentation/data\\_sheets/ds100.pdf](http://www.xilinx.com/support/documentation/data_sheets/ds100.pdf).
- Yig10a. Ozan Yigit. Hash Functions, April 2010. <http://www.cse.yorku.ca/~oz/hash.html>.
- Yig10b. Ozan Yigit. sdbm - Substitute DBM or Berkeley ndbm for Every UN\*X[1] Made Simple, April 2010. <http://cpansearch.perl.org/src/JESSE/perl-5.12.0-RC5/ext/SDBM.File/sdbm/README>.

## *Vita*

Before attending the Air Force Institute of Technology, Brennon Thomas previously spent three years on active duty as an Air Force Communications Officer at the former Air Force Communications Agency (currently the Air Force Network Integration Center) at Scott Air Force Base, Illinois. He received a Bachelor of Science degree in Electrical Engineering from Rensselaer Polytechnic Institute in 2005. He also commissioned through the Air Force Reserve Officer Training Corps program at Rensselaer Polytechnic Institute in 2005.

Permanent address: 2950 Hobson Way  
Air Force Institute of Technology  
Wright-Patterson AFB, OH 45433

<b>REPORT DOCUMENTATION PAGE</b>					<i>Form Approved</i> <b>OMB No. 0704-0188</b>	
The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. <b>PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.</b>						
<b>1. REPORT DATE</b> (DD-MM-YYYY) 17-06-2010		<b>2. REPORT TYPE</b> Master's Thesis			<b>3. DATES COVERED</b> (From — To) Sept 2008 — June 2010	
<b>4. TITLE AND SUBTITLE</b>  Performance Evaluation of a Field Programmable Gate Array-Based System for Detecting and Tracking Peer-to-Peer Protocols on a Gigabit Ethernet Network					<b>5a. CONTRACT NUMBER</b>  <b>5b. GRANT NUMBER</b>  <b>5c. PROGRAM ELEMENT NUMBER</b>  <b>5d. PROJECT NUMBER</b> N/A <b>5e. TASK NUMBER</b>  <b>5f. WORK UNIT NUMBER</b>  	
<b>6. AUTHOR(S)</b>  Brennon D. Thomas					<b>8. PERFORMING ORGANIZATION REPORT NUMBER</b>  AFIT/GCO/ENG/10-20	
<b>7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)</b> Air Force Institute of Technology Graduate School of Engineering and Management (AFIT/EN) 2950 Hobson Way WPAFB OH 45433-7765					<b>10. SPONSOR/MONITOR'S ACRONYM(S)</b> 688th Information Operations Wing	
<b>9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)</b> 688th Information Operations Wing Attn: Mr. Robert J. Kaufman 102 Hall Boulevard, Suite 345 San Antonio, TX 78243 DSN 969-5114; robert.kaufman@us.af.mil					<b>11. SPONSOR/MONITOR'S REPORT NUMBER(S)</b>	
<b>12. DISTRIBUTION / AVAILABILITY STATEMENT</b>  Approval for public release; distribution is unlimited.						
<b>13. SUPPLEMENTARY NOTES</b>						
<b>14. ABSTRACT</b> The TRacking and Analysis for Peer-to-Peer 2 (TRAPP-2) system is developed on a Xilinx ML510 FPGA. The goals of this research are to evaluate the performance of the TRAPP-2 system as a solution to detect and track malicious packets traversing a gigabit Ethernet network. The TRAPP-2 system detects a BitTorrent, Session Initiation Protocol (SIP), or Domain Name System (DNS) packet, extracts the payload, compares the data against a hash list, and if the packet is suspicious, logs the entire packet for future analysis. Results show that the TRAPP-2 system captures 95.56% of BitTorrent, 20.78% of SIP INVITE, 37.11% of SIP BYE, and 91.89% of DNS packets of interest while under a 93.7% network utilization (937 Mbps). For another experiment, the contraband hash list size is increased from 1,000 to 131,072,000 unique items. The experiment reveals that each doubling of the hash list size results in a mean increase of approximately 16 central processing unit cycles. These results demonstrate the TRAPP-2 system's ability to detect traffic of interest under a saturated network utilization while maintaining large contraband hash lists.						
<b>15. SUBJECT TERMS</b>  FPGA, gigabit, BitTorrent, SIP, VoIP, DNS, peer-to-peer						
<b>16. SECURITY CLASSIFICATION OF:</b>			<b>17. LIMITATION OF ABSTRACT</b>		<b>18. NUMBER OF PAGES</b>	
a. REPORT	b. ABSTRACT	c. THIS PAGE	UU		149	
U	U	U	<b>19a. NAME OF RESPONSIBLE PERSON</b> Dr. Barry E. Mullins			
						<b>19b. TELEPHONE NUMBER</b> (include area code) (937) 255-3636 x7979; barry.mullins@afit.edu